Titre: Title:	Affectation de cellules à des commutateurs par programmation par contraintes
Auteur: Author:	Grâce Amoussou
Date:	2001
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Amoussou, G. (2001). Affectation de cellules à des commutateurs par programmation par contraintes [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/6970/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/6970/
Directeurs de recherche: Advisors:	
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

AFFECTATION DE CELLULES À DES COMMUTATEURS PAR PROGRAMMATION PAR CONTRAINTES

GRÂCE AMOUSSOU
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
Avril 2001

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

AFFECTATION DE CELLULES À DES COMMUTATEURS PAR LA PROGRAMMATION PAR CONTRAINTES

présenté par: Grâce Amoussou

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. CONAN Jean, Ph.D., Président

M. PIERRE Samuel, Ph.D., membre et directeur de recherche

M. PESANT Gilles, Ph. D., membre et codirecteur de recherche

M. SORIANO Patrick, Ph.D., membre

à mon père ...

REMERCIEMENTS

Plusieurs personnes m'ont apporté leur soutien pour la réalisation de ces travaux de recherche.

De façon particulière, j'aimerais remercier mon directeur de recherche M. Samuel Pierre sans qui je n'aurai pas eu cette opportunité. C'est l'occasion pour moi de lui exprimer ma reconnaissance pour le choix du sujet, son aide financière, sa patience mais surtout pour les nombreux conseils qui m'ont soutenu et guidé au cours de cette recherche.

Je remercie également M. Gilles Pesant, mon codirecteur, pour son aide financière et pour avoir partagé avec moi sa connaissance du sujet.

Je remercie Joël, mon frère qui m'a permis de réaliser mes études universitaires, mes parents, frères et sœurs pour leur soutien moral durant toutes ces années d'études.

Enfin, je remercie tous mes collègues du LARIM (Laboratoire de recherche en Réseautique et Informatique Mobile) et du CRT (Centre de Recherche sur les Transports) pour leurs contributions variées au contenu de ce mémoire et en particulier pour l'ambiance de travail chaleureuse.

RÉSUMÉ

Les réseaux de communications personnelles *(RCP)* offrent divers types de services comme les données, la voix, la vidéo, le courrier électronique sur un même support non filaire. La gestion de la mobilité des usagers à l'intérieur de tels réseaux entraîne plusieurs problèmes jusque là inexistants dans les réseaux commutés publics. Dans la résolution de certains de ces problèmes, qui se trouvent être de nature NP-difficiles, on a souvent recours à des méthodes et techniques utilisées en optimisation combinatoire.

Ce mémoire porte sur l'adaptation des techniques de la programmation par contraintes pour la résolution d'un problème bien connu dans les RCP et qui est celui de l'affectation de cellules à des commutateurs. Il s'agit en fait de déterminer une affectation des différentes cellules du réseau à des commutateurs (dont les localisations sont fixes et connues), qui minimiserait une fonction de coûts composée des coûts de relève entre toutes les cellules d'une part, et des coûts de liaison entre cellules et commutateurs d'autre part. De plus, toute solution réalisable devrait respecter la limite de capacité de chaque commutateur. Le nombre d'affectations à considérer peut entraîner une explosion combinatoire.

La programmation par contraintes permet de simplifier la résolution de problèmes combinatoires complexes. L'idée de base consiste à concevoir des techniques de filtrage efficaces pour réduire l'espace de recherche. Ainsi, dans notre adaptation, nous avons défini une modélisation des inconnues du problème sous forme de contraintes qui réduit considérablement l'espace de recherche. L'algorithme utilise la technique du "Contraindre et Générer" pour imposer le respect des contraintes; par la suite, la méthode de "limitation et exploration" (Branch & Bound) est appliquée pour trouver la meilleure solution.

Comme contributions majeures de ce mémoire, nous avons pu définir une contrainte sur la borne inférieure du coût de relève qui permet d'éliminer de la recherche plusieurs solutions non réalisables. La mise au point d'un ensemble de stratégies de

sélection des variables, spécifiques à ce problème d'affectation, nous a permis de diriger de manière dynamique la recherche et d'aboutir à de bons résultats.

Dans le but d'évaluer la performance de l'algorithme par rapport aux autres heuristiques adaptées à ce même type de problème, nous avons effectué une série de tests. Malgré sa simplicité, notre adaptation fournit des solutions exactes aux réseaux de taille moyenne et donne des solutions qui se comparent avantageusement aux autres méthodes pour des réseaux de grande taille.

Les résultats obtenus montrent que la définition d'une contrainte sur la borne inférieure du coût de relève permet de réduire de manière considérable le domaine des valeurs prises par les variables de modélisation. De plus, les stratégies de choix de variables dans l'algorithme de "Branch & Bound", varient suivant les types de problème; elles sont plus efficaces lorsqu'elles sont dynamiques et tiennent compte des informations disponibles à chaque étape de la recherche. Enfin, même si le temps de calcul peut parfois être onéreux pour certaines tailles de réseaux, les solutions obtenues pour les problèmes de taille moyenne sont optimales et peuvent servir comme critère pour une évaluation de la distance des autres heuristiques par rapport à la meilleure solution.

ABSTRACT

Personal Communication Services (*PCS*) networks offer a lot of services like transmission of voice, video and e-commerce over wireless support. The fact that users inside those networks are free to move causes a lot of challenges to the providers of those kinds of services. Most of the methods used to solve those problems are from operations research.

Our thesis discusses a very important problem of cell assignment to switches in cellular mobile networks. It can be summarized as a search for an optimal assignment of cells to switches in order to minimize the total cost composed of the handoff cost between cells and the linking cost between cells and switches. We propose here an algorithm based on constraint programming for this problem. The choice of this method is motivated by its active use of constraints in the search for solutions, which in turn leads to the reduction of the search space and of the complexity of the problem. The COP (Constraint Optimization Problem) used here is based on "Branch & Bound" techniques.

The principal contributions of this thesis are:

- A Constraint Programming (*CP*) modeling of the problem of assigning cells to switches;
- The definition of a lower bound on the total handoff cost between cells;
- The development of a new daemon and search strategies. Daemons help to handle dynamically the relations between the constrained variables, as the strategies direct the search. The result is an efficient way of finding the best solutions.

In order to evaluate the efficiency of our method, we compared it with other heuristics that have been adapted to the same problem. Results indicated that our algorithm leads to optimal solutions for medium - sized networks and can deliver a satisfactory solution for large - scale networks.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES SIGLES ET ABRÉVIATIONS

AG	Algorithme Génétique
BB	Branch &Bound
BI	Borne Inférieure
BSC	Base Station Controller
BST	Base Station Transceiver
CLP	Constraint Logic Programming
COP	Constraint Optimisation Problem
CSP	Constraint Satisfaction Problem
MSC	Mobile Switching Center
PC	Programmation par Contraintes
PCS	Personal Communication Services
RC	Recuit Simulé
RCP	Réseaux de communications personnelles
RT	Recherche Taboue

CHAPITRE 1
INTRODUCTION

Nous assistons ces derniers temps à une prolifération des services offerts par les systèmes de communications personnelles. Les transferts de données, d'image, et de vidéo sont autant de services accessibles aux utilisateurs de ces types de réseaux dont le nombre ne cesse de croître. Il va sans dire que tous ces changements entraîneront, si ce n'est déjà le cas, une restructuration dans la conception et dans la gestion des futurs réseaux de communications mobiles. Parmi les problèmes répertoriés et qui demeurent incontournables pour l'administration efficace de ces futurs réseaux se trouve celui de l'affectation des cellules à des commutateurs, qui fait l'objet de ce mémoire. Dans ce chapitre, nous allons définir dans un premier temps les concepts fondamentaux utilisés dans les réseaux de communications personnelles afin de préciser notre problématique de recherche. Nous exposerons par la suite nos objectifs de recherche et les principales contributions escomptées et enfin nous esquisserons le plan du mémoire.

1.1 Définitions et concepts de base

Les services de communications personnelles sont offerts sur des *réseaux de communications personnelles* (*RCP*). Le territoire couvert ou la *zone de couverture* que peuvent desservir de tels réseaux est généralement découpée en de petites surfaces géographiquement limitées et communément appelées *cellules*. Celles-ci sont souvent représentées par des hexagones dont le rayon varie de quelques centaines de mètres à quelques kilomètres au maximum. À l'intérieur de chacune de ces cellules se trouve un *sous-système radio* constituant une *station de base* ou *BST* (*Base Station Transceiver*) qui s'occupe des transmissions radio sur la cellule. Intégrés à la station de base, des canaux de signalisation vont permettre à l'abonné de communiquer avec la *BST* et vice versa. Les stations de base sont à leur tour reliées à des *contrôleurs de station de base* ou *BSC* (*Base Station Controller*). C'est ce sous-système qui sert donc *d'interface radio* entre chaque terminal mobile et le réseau lui-même. Le sous-système réseau est à son tour constitué de *commutateurs* ou *MSC* (*Mobile Switching Center*) installés à

l'intérieur de quelques-unes des cellules choisies de manière stratégique. Le rôle d'un commutateur est d'assurer l'interconnexion des différentes cellules du réseau mobile entre elles et aussi avec les autres réseaux de télécommunications.

Les terminaux mobiles sont en général utilisés en déplacement. Pour éviter des interférences, deux cellules contiguës n'utilisent pas les mêmes canaux radio. La transmission doit donc changer de canal chaque fois que le mobile passe d'une cellule à une autre. Ce processus de transfert automatique de la communication d'une station de base à une autre est appelé *relève* (*handover* ou *handoff*). Concrètement, le système cellulaire contrôle en permanence la puissance du signal entre le mobile et la station de base dans laquelle il se situe. Dès que la puissance tombe sous un niveau donné, le système attribue automatiquement une nouvelle cellule au mobile. Ce transfert de cellules peut entraîner un changement de commutateur, auquel cas des opérations de mise à jour sont réalisées et on parle de *relève complexe*, sinon on a une *relève simple* ne faisant intervenir qu'un même et unique commutateur.

1.2 Éléments de la problématique

Dans un *RCP*, les usagers peuvent être en déplacement à l'intérieur d'une surface géographique donnée. La liaison avec le réseau est assurée par interface radio, c'est-à-dire que chaque cellule est munie d'une antenne qui lui permet de communiquer avec les usagers sur différentes fréquences. La communication avec chaque usager est prise en charge par l'une des cellules les plus proches, choisie en tenant compte du niveau du signal reçu. Généralement, on définit un seuil au-delà duquel on considère que la puissance reçue est assez importante pour être prise en compte par une cellule. Les informations échangées par les usagers sont à leur tour gérées par le commutateur qui dessert cette cellule. Celle-ci est donc desservie par un seul commutateur à la fois. Lorsque différentes cellules (situées à proximité) reçoivent toutes des niveaux du signal supérieur au seuil, seule celle ayant le niveau le plus élevé assurera la prise en charge de l'usager. De ce fait, la communication sera relayée par le commutateur auquel ladite cellule est affectée. Plusieurs cas peuvent se présenter lors du déplacement du mobile

d'une cellule *A*, reliée à un commutateur *X*, vers une autre cellule *B* ayant un plus fort niveau du signal:

1) Les cellules *A* et *B* sont contrôlées par un même commutateur, dans ce cas la relève est simple et ne nécessite pas des opérations de mise à jour de la base de données de l'usager. Seul le commutateur *X* intervient dans cette opération.

2) La cellule *B* est contrôlée par un commutateur *Y* différent de *X*. Dans ce cas, plusieurs informations sont échangées entre les deux commutateurs pour mettre à jour la base de données du réseau (localisation de l'usager, type d'appels, etc.). De plus, il peut arriver que certaines opérations, comme la facturation, continuent d'être effectuées par le commutateur *X*. On aura ainsi une connexion de l'usager au commutateur *Y*, puis au commutateur *X* et enfin au réseau. On parle de relève complexe dont le coût est très élevé comparativement à une relève simple.

La relève complexe constitue une opération sollicitant beaucoup de ressources de la part du réseau et dont il convient de réduire autant que possible le coût. De ce fait, il serait souhaitable d'établir une fréquence des relèves entre les différentes cellules, afin de pouvoir regrouper celles échangeant le plus, sous le contrôle d'un même commutateur. Ces différentes considérations sont à la base du problème d'affectation de cellules à des commutateurs qui peut être énoncé de la manière suivante:
Étant donné un ensemble de cellules et de commutateurs de capacités (*Erlang*) finies, trouver une affectation des cellules à ces commutateurs qui minimiserait le coût total constitué du coût de liaison entre cellules et commutateurs d'une part, et du coût de relève entre les cellules d'autre part. La résolution de ce problème doit donc prendre en compte les facteurs suivants: la topologie du réseau, la capacité des commutateurs et le volume des appels échangés par unité de temps dans chacune des cellules.

Plusieurs méthodes, pour la plupart heuristiques, ont été proposées pour sa résolution. Merchant et Sengupta (1994) ont résolu le problème suivant deux schémas

d'affectation. Le premier schéma impose l'unicité de l'affectation dans la configuration du réseau: c'est la domiciliation simple. Le second schéma considère qu'indépendamment du trafic, une affectation efficace à un moment de la journée peut l'être moins à un autre moment. On doit alors réaliser une domiciliation double, c'est-à-dire permettre qu'une cellule puisse être affectée à deux commutateurs différents auxquels elle sera reliée suivant les moments de la journée. Dans l'un ou l'autre cas, le problème à résoudre demeure très complexe et nécessite une approche heuristique.

1.3 Objectifs de recherche et principales contributions escomptées

Notre objectif dans ce mémoire est d'appliquer les méthodes de la programmation par contraintes *(PC)* à ce problème. Pour ce faire, nous introduirons une nouvelle modélisation du problème. Celle-ci utilise les variables, ainsi que les contraintes qu'elles doivent satisfaire, de manière active, pour aboutir à une bonne réduction de l'espace de recherche. Nous essayerons par la suite d'appliquer plusieurs jeux de démons pouvant permettre de mieux propager les contraintes du problème sur les différentes variables utilisées. De plus, diverses stratégies de recherche seront exploitées pour réduire le temps de calcul, surtout pour des problèmes de grande taille. L'utilisation de ces stratégies permettra de se rendre compte très tôt des échecs et de limiter la zone de recherche à des parties de l'arbre pouvant déboucher sur de bonnes solutions.

Les principales contributions escomptées sont les suivantes:

1. La modélisation sous forme *PC* du problème d'affectation de cellules à des commutateurs, l'utilisation des variables ensemblistes parmi les variables de contraintes, la définition de différents démons pour établir des liens dynamiques entre les différentes variables, à travers leurs différents domaines, la définition d'une borne inférieure du coût de relève;
2. La détermination d'une solution exacte en des temps de calcul raisonnables pour les réseaux de taille moyenne ;

3. La définition de nouvelles stratégies de recherche pour optimiser le temps de calcul.

Il en résultera un outil de résolution de ce problème basé sur la programmation par contraintes. Le choix de cette méthode est justifié par sa simplicité et son utilisation efficace des contraintes dans la résolution des problèmes du même type et reconnus comme NP-difficiles.

1.4 Plan du mémoire

Le mémoire est divisé en cinq chapitres. Après ce premier chapitre d'introduction, le chapitre 2 présente une formulation sous forme de programmation en nombres entiers du problème d'affectation, ainsi qu'une revue des méthodes qui lui ont été appliquées. Le chapitre 3 décrit la méthode de programmation par contraintes que nous nous proposons d'appliquer dans ce mémoire. Le chapitre 4 expose les détails d'implémentation, de mise en œuvre de la méthode et une analyse des résultats. Enfin, le chapitre 5 présente une synthèse des travaux mettant en évidence les principaux résultats obtenus et leurs limitations ainsi qu'une indication des recherches futures.

CHAPITRE 2
AFFECTATION DE CELLULES À DES COMMUTATEURS

L'affectation des cellules à des commutateurs dans les réseaux de communications personnelles *(RCP)* peut être considérée comme un problème de gestion de ressources qui se pose dans la phase de conception de ces réseaux. Nous allons, dans ce chapitre, présenter de manière succincte ce problème. Nous commencerons par une analyse de l'architecture et des caractéristiques des réseaux cellulaires. Par la suite, nous présenterons une formulation mathématique du problème d'affectation, ce qui nous amènera à examiner quelques travaux effectués en recherche opérationnelle qui traitent de certains de ces aspects. Enfin, nous passerons en revue certaines méthodes proposées pour la résolution de ce problème.

2.1 Architecture et caractéristiques des RCP

Les réseaux mobiles de troisième génération, souvent désignés par réseaux de communications personnelles et les réseaux cellulaires de deuxième génération possèdent une architecture très proche l'une de l'autre. Cependant, contrairement aux systèmes de deuxième génération, les systèmes de troisième génération fournissent des services de types variés et ayant des débits différents. Dans cette section, nous allons présenter l'architecture de ces réseaux ainsi que les différentes caractéristiques techniques que l'on doit prendre en compte pour affecter des cellules à des commutateurs dans ce contexte.

2.1.1 Architecture des RCP

Les *RCP* désignent l'ensemble de tous les systèmes de télécommunications offrant des services de communication tels la voix, les données numériques, le multimédia sur des supports de transmission non filaires et à des usagers mobiles. Chaque abonné d'un tel système est identifié au travers d'une carte à mémoire, la carte

SIM (*Module d'Identification de l'Abonné*) qui lui permet de se connecter au réseau pour bénéficier des services auxquels il est abonné et ce, quelle que soit sa localisation.

L'architecture des *RCP* est basée essentiellement sur celle des réseaux cellulaires. On retrouve une décomposition de toute la superficie à desservir en petites zones géographiques souvent modélisées par des formes hexagonales et contiguës assurant une couverture complète de la zone. Ces zones sont communément appelées *cellules*. À l'intérieur de chacune de ces dernières se trouve une station émettrice de base (*BST* : *Base Station Transceiver*) qui sert *d'interface radio* entre chaque mobile et le réseau d'une part, et entre le réseau et les abonnés d'autre part. La *BST* constitue avec le contrôleur de station de base (*BSC* : *Base Station* Controller) le *sous-système radio* dont la principale fonction est de prendre en charge la transmission et la signalisation entre les différents utilisateurs d'une cellule. Le *sous-système réseau* est à son tour constitué des différents centres de commutation du service mobile (*MSC* : *Mobile Switching Center*) et son rôle est d'assurer l'*interconnexion* des différentes stations de base non seulement entre elles mais aussi avec les autres types de réseaux comme le réseau public. De par leur fonction, les nœuds de commutation que sont les MSC représentent des points centraux et sont placés à l'intérieur de certaines cellules, choisies de manière stratégique. Chaque commutateur gère un certain nombre de stations de base et dispose d'une *capacité* maximale fixée. Celle-ci est souvent exprimée en termes de *volume d'appels*, que reçoit le commutateur des différentes cellules qui lui sont affectées. Ces MSC sont par la suite reliés entre eux, comme l'illustre la Figure 2.1. On introduit parfois la notion de *charge partagée (balancing loading)* qui consiste à répartir le trafic entre les commutateurs.

2.1.2 Caractéristiques des RCP

Lorsqu'un utilisateur est à l'intérieur du réseau, son terminal est raccordé à une des stations de base (ou cellules) en fonction de la puissance du signal qu'il reçoit. Dans son déplacement, si cette puissance tombe au-dessous d'un certain seuil, certaines opérations sont mises en œuvre pour la relayer par une nouvelle cellule.

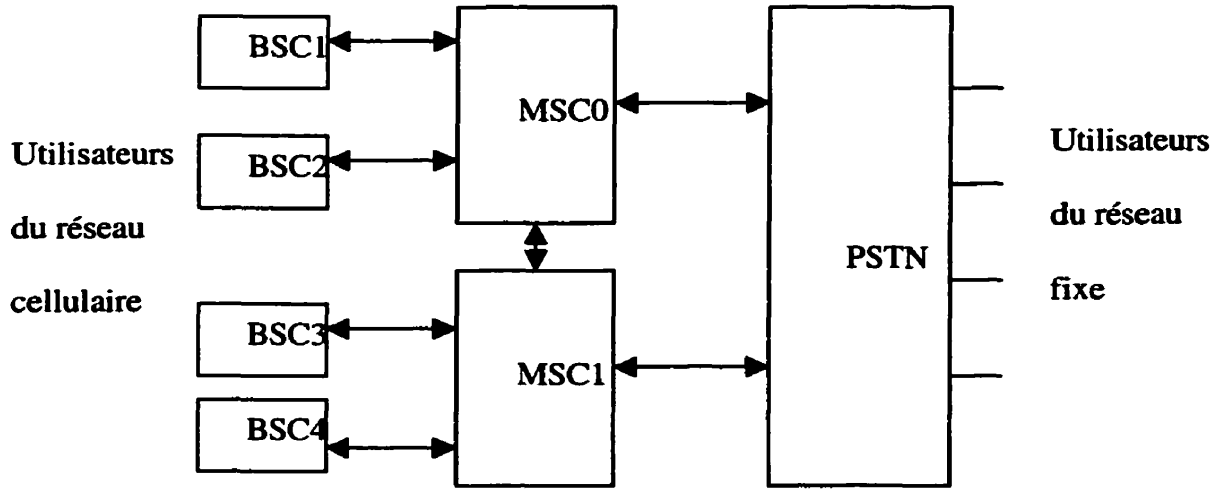

Figure 2. 1 Architecture d'un réseau cellulaire

Le processus qui permet d'initier et d'effectuer un changement de cellule, avec les mises à jour que cela nécessite constitue la *relève* (*handoff* ou *handover*). Étant donné la nature des signaux transmis sur les réseaux, la relève doit s'effectuer de manière transparente à l'unité mobile. On distingue généralement deux types de relève: la relève complexe et la relève simple. La relève est dite complexe dans le cas où elle s'accompagne d'un changement de *MSC*, c'est-à-dire que la nouvelle cellule est contrôlée par un commutateur autre que la cellule quittée. Dans ce cas, plusieurs opérations de mise à jour sont effectuées et celles-ci demandent une forte consommation des ressources du réseau. Par contre, lorsque les deux cellules sont reliées à un même commutateur, aucune mise à jour n'est réalisée et on parle de *relève simple*. La Figure 2.2 illustre la relève à l'intérieur d'un réseau cellulaire. Si l'usager passe de la cellule C_1 à la cellule C_3, seul le commutateur MSC_2 est concerné et aucune mise à jour n'a besoin d'être faite: c'est donc une relève simple. Les informations de signalisation et de facturation continuent alors d'être gérées par une même entité. Mais, pour un

déplacement du mobile de la cellule C_1 vers la cellule C_0, les deux entités du réseau que sont les commutateurs MSC_3 et MSC_2 entrent en dialogue pour mettre à jour leur base de données à travers les enregistreurs de localisation. C'est un cas de relève complexe. Il peut arriver que le MSC_2 transmette toutes les informations au MSC_3 qui se charge alors d'assurer le relais. Mais, dans certaines situations par exemple où le commutateur MSC_2 est chargé de la facturation, il demeurera en contact avec le mobile à travers le commutateur MSC_3 et ce, jusqu'à la fermeture de la connexion.

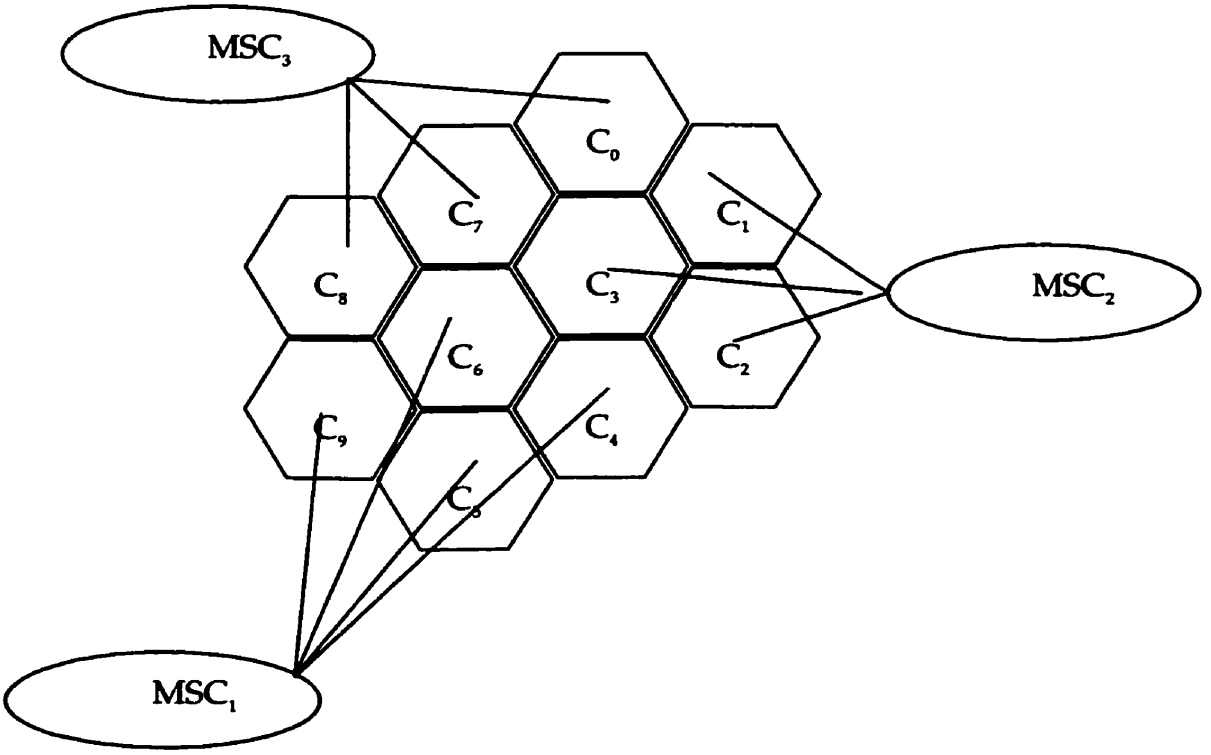

Figure 2. 2 Relève dans un réseau cellulaire

Chaque opération de relève nécessite des ressources du réseau: mise à jour des données de localisation dans les bases de données, utilisation de protocoles de communications entre *MSC*, etc. On doit trouver des mécanismes pour réduire au mieux les coûts qu'elle entraîne. Un des moyens consiste à optimiser l'affectation de cellules

aux commutateurs. Dans ce cas, on peut tenir compte des fréquences d'appel ou *patron d'appel* entre les différentes cellules du réseau. Deux cellules échangeant des quantités considérables d'informations seront affectées à un même commutateur pour minimiser le coût de relève. De plus, on peut considérer aussi une domiciliation simple ou double des cellules (Merchant et Sengupta, 1995). On parle de *domiciliation simple* lorsqu'une cellule ne peut être reliée qu'à un seul commutateur. La *domiciliation est double* quand on peut connecter une cellule à au plus deux commutateurs et ce, suivant les moments de la journée. Les deux commutateurs auxquels la cellule est reliée sont alors actifs de manière alternative en fonction des périodes de la journée (un patron pour la matinée et un autre pour le soir). Ces différents concepts caractérisent les réseaux mobiles et interviennent dans le problème d'affectation de cellules à des commutateurs dans les *RCP*.

2.2 Formulation du problème d'affectation de cellules

Le problème d'affectation de cellules à des commutateurs peut être modélisé suivant plusieurs approches. Merchant et Sengupta (1994, 1995) l'ont formulé comme un problème de programmation en nombres entiers: minimiser une fonction de coût, suivant un schéma d'affectation tout en respectant les contraintes sur la capacité de chaque commutateur. Samadi et Wong (1992) se sont plutôt intéressés à la minimisation du nombre de mises à jour des données de localisation. Dans ce qui suit, nous présenterons le premier modèle (basé sur le coût) réalisé suivant deux schémas d'affectation: simple et double.

2.2.1 Modélisation suivant la domiciliation simple

On suppose que le réseau dispose de n cellules et de m commutateurs dont les emplacements sont connus. À chaque commutateur doit être affecté un ensemble de cellules suivant le volume des appels qu'il peut gérer. Pour chaque paire de cellules i et j ($i \neq j$), on définit les coûts suivants:

H_{ij} – Coût par unité de temps d'une relève simple entre les cellules i et j;

H'_{ij} – Coût par unité de temps d'une relève complexe entre les cellules i et j;

C_{ik} – Coût d'amortissement provenant du câblage entre la cellule i et son commutateur k.

Soient λ_i le volume d'appels par unité de temps reçu par la cellule i ($1 \leq i \leq n$) et M_k la capacité du commutateur k ($1 \leq k \leq m$).

L'objectif est de trouver une affectation des cellules aux commutateurs qui minimise la somme totale des coûts de liaison et de relève et respecte la contrainte de capacité limitée de chaque *MSC*. Pour décrire le problème, on introduit les $n \times m$ variables binaires suivantes:

$$X_{ik} = \begin{cases} 1 & \text{si la cellule est reliée au commutateur k} \\ 0 & \text{sinon} \end{cases}$$

On s'intéresse alors à exprimer le problème à l'aide de ces variables de manière à satisfaire les contraintes qui y sont posées. Tout d'abord, au niveau des cellules, chacune d'elles doit être assignée à un et un seul commutateur. Cette condition peut être représentée par la relation:

$$\sum_{k=1}^{m} X_{ik} = 1 \qquad \text{pour } i = 1, \ldots, n \tag{2. 1}$$

D'autre part, si C_{ik} désigne le coût d'amortissement de la liaison entre la cellule i et le commutateur k, on peut exprimer le coût total de liaison entre toutes les cellules et les commutateurs auxquels elles sont reliées par la relation:

$$\sum_{i=1}^{n} \sum_{k=1}^{m} C_{ik} X_{ik} \tag{2. 2}$$

Pour représenter le coût total induit par les opérations de relèves simples et complexes, on introduit de plus les variables complémentaires suivantes:

$$Z_{ijk} = X_{ik} . X_{jk} \qquad \text{pour i, j} = 1, \ldots, n \text{ et k} = 1, \ldots, m \tag{2. 3}$$

Ces variables permettent en effet de formuler mathématiquement le fait que deux cellules *i* et *j* soient affectées à un même commutateur k par la propriété suivante:

$$Z_{ijk} = \begin{cases} 1 & \text{si i et j sont connectées au commutateur k} \\ 0 & \text{sinon} \end{cases} \qquad (2.4)$$

Soit maintenant :

$$Y_{ij} = \sum_{k=1}^{m} Z_{ijk} \quad \text{pour i, j} = 1, \ldots, n \text{ et } i \neq j \qquad (2.5)$$

Y_{ij} prend la valeur 1 si les cellules *i* et *j* sont connectées à un même commutateur et est égale à 0 si elles sont reliées à des commutateurs différents.

Le coût des relèves simples et complexes par unité de temps s'exprime par :

$$f_{hs} = \sum_{i=1}^{n}\sum_{j=1}^{n} H_{ij} Y_{ij} \text{ pour la relève simple} \qquad (2.6)$$

$$f_{hc} = \sum_{i=1}^{n}\sum_{j=1}^{n} H'_{ij}(1-Y_{ij}) \text{ pour la relève complexe} \qquad (2.7)$$

La fonction objective globale composée de chacun des coûts prédéfinis s'écrit donc:

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} C_{ik} X_{ik} + \sum_{i=1}^{n}\sum_{j=1}^{n} H_{ij} Y_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n} H'_{ij}(1-Y_{ij}) \qquad (2.8)$$

Il s'agit alors de minimiser la fonction *f* sous les contraintes suivantes:

$$\sum_{i=1}^{n} X_{ik} = 1 \text{ pour } i = 1, \ldots, n$$

$$X_{ik} = 0 \text{ } ou \text{ } 1 \text{ pour } i = 1, \ldots, n \text{ et } k = 1, \ldots, m$$

$$Z_{ijk} = X_{ik} . X_{jk} \quad \text{pour i, j} = 1, \ldots, n \text{ et k} = 1, \ldots, m$$

$$Y_{ij} = \sum_{k=1}^{m} Z_{ijk} \text{ pour i, j} = 1, \ldots, n \text{ et } i \neq j$$

De plus, on a la contrainte imposée par la capacité de chaque commutateur et suivant laquelle le volume total d'appels engendré par toutes les cellules liées au commutateur k ne doit pas dépasser la capacité maximum de ce commutateur. Celle-ci se traduit par:

$$\sum_{i=1}^{n} \lambda_i X_{ik} \leq M_k \quad pour \quad k=1,2,...,m \qquad (2.9)$$

Pour simplifier la fonction f de la relation (2.8), on peut négliger le coût des relèves simples devant celui des relèves complexes qui utilisent plus de ressources. De ce fait, si on pose:

$$h_{ij} = H'_{ij} - H_{ij} \approx H'_{ij}$$

qui représente le coût réduit par unité de temps d'une relève complexe entre les cellules i et j de sorte que f peut s'écrire:

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} C_{ik} X_{ik} + \sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n} H_{ij} Y_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n} (h_{ij} + H_{ij})(1 - Y_{ij})$$

qui est équivalent à:

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} C_{ik} X_{ik} + \sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n} H_{ij} + \sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n} h_{ij}(1 - Y_{ij})$$

Du fait que la sommation :

$$\sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n} H_{ij}$$

est une constante, le problème initial peut s'écrire sous la forme suivante :

Minimiser

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} C_{ik} X_{ik} + \sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n} - H'_{ij} Y_{ij} \qquad (2.10)$$

sous les contraintes:

$$X_{ik} = \begin{cases} 1 & \text{si la cellule est reliée au commutateur k} \\ 0 & \text{sinon} \end{cases}$$

$$\sum_{i=1}^{n} X_{ik} = 1 \text{ pour } i = 1, \ldots, n$$

$$Z_{ijk} = X_{ik} X_{jk} \text{ pour } i, j = 1, \ldots, n \text{ et } k = 1, \ldots, m$$

$$Y_{ij} = \sum_{k=1}^{m} Z_{ijk} \text{ pour } i, j = 1, \ldots, n \text{ et } i \neq j$$

$$Y_{ij} = \begin{cases} 1 & \text{si i et j sont reliées au même commutateur} \\ 0 & \text{sinon} \end{cases}$$

$$\sum_{i=1}^{n} \lambda_i X_{ik} \leq M_k \quad pour \quad k=1,2,\ldots,m$$

Pour ramener le problème à un problème de programmation en nombres entiers, Merchant et Sengupta ont proposé de remplacer la contrainte non linéaire (2.3) par un ensemble de contraintes équivalentes:

$$Z_{ijk} \leq X_{ik}$$
$$Z_{ijk} \leq X_{jk}$$
$$Z_{ijk} \geq X_{ik} + X_{jk} - 1$$
$$Z_{ijk} \geq 0.$$

2.3 Caractérisation du problème d'affectation

Tel que formulé précédemment, le problème d'affectation de cellules aux commutateurs peut être ramené à plusieurs types de problème largement étudiés en recherche opérationnelle tels le problème de transport ou de localisation de concentrateurs (Skorin-Kapov et al., 1994; Klincewicz, 1988; O'Kelly, 1987) et celui de partitionnement de graphes (Kerninghan, 1970; Sanchis, 1989). Leur résolution par une méthode énumérative conduit généralement à une croissance exponentielle du temps

d'exécution. En conséquence, on recherche une solution plutôt proche de l'optimum, en développant des heuristiques ou méta-heuristiques de résolution pour ces types de problèmes reconnus NP-difficiles. Dans ce qui suit, nous présenterons quelques-uns de ces problèmes ainsi que leurs caractéristiques par rapport au problème d'affectation.

2.3.1 Problème de transport

Dans un problème de transport, on dispose de deux ensembles disjoints. En particulier pour le problème de localisation de concentrateurs, on dispose de n nœuds dont les emplacements sont connus. Chacun de ces nœuds peut échanger du trafic avec les autres nœuds du réseau. Le but est de localiser parmi les n nœuds, p concentrateurs et de leur affecter les n-p nœuds restants, sous des contraintes liées à la capacité des différents concentrateurs. Le problème d'affectation est donc un cas particulier de celui de transport. Dans les deux cas, on dispose de deux ensembles disjoints et on essaie d'établir une correspondance entre les éléments de ces ensembles. Ainsi, dans le problème d'affectation, les p nœuds concentrateurs peuvent être représentés comme des commutateurs et les n-p restants comme des cellules. Étant donné que l'emplacement des commutateurs est connu, on cherchera alors à résoudre uniquement la partie affectation. Notons toutefois qu'il existe quelques différences entre les deux problèmes:

- Le problème de localisation de concentrateurs est un problème mixte dans ce sens qu'il peut faire intervenir des valeurs fractionnaires d'un nœud i à un autre j. Cela veut dire qu'un même flot peut être partagé sur deux liaisons, contrairement au problème d'affectation de cellules qui est complètement entier.
- Dans le problème de localisation, on a des concentrateurs dont la localisation n'est pas encore fixée. Il s'agit alors dans bien des cas de déterminer d'abord un choix pour la localisation et ensuite de faire l'affectation des nœuds restants. Par contre, dans le problème d'affectation, les emplacements des commutateurs sont déjà fixés et connus et l'on ne résout que le problème d'affectation qui

n'en reste pas moins complexe. Ainsi, il n'y a pas de coût lié à la localisation des commutateurs.

- On a une contrainte supplémentaire dans le problème d'affectation de cellules à des commutateurs qui est fonction de la capacité de ces derniers et qui n'apparaît pas clairement dans le problème de localisation de concentrateurs. Celle-ci exprime que chaque commutateur peut supporter un nombre limité de volumes d'appels par unité de temps provenant de l'ensemble des cellules dont il a la charge.

2.3.2 Problème de partitionnement de graphes

Le partitionnement de graphes est un problème souvent présent lors du partage des ressources dans le domaine des technologies de l'information. Il peut survenir dans la construction des réseaux de télécommunications ou en calcul parallèle où l'on doit répartir différentes tâches entre plusieurs processeurs.

Soit G un graphe, le problème de partitionnement de graphe consiste à diviser l'ensemble N des nœuds du graphe en des sous-ensembles de cardinalité inférieure à un nombre maximal donné, de manière à minimiser la somme totale des coupes. Une coupe représentant tous les arcs du graphe ayant leurs extrémités dans des sous-ensembles différents. Un réseau peut être représenté sous la forme d'un graphe où les nœuds représentent les terminaux du réseau et les arcs les liaisons entre ces différents terminaux. Pour interconnecter les différents nœuds du réseau, on réalise un partitionnement de ses nœuds en sous-graphes reliés par un nombre minimum de liaisons mais assez robustes pour éviter des congestions. Le problème d'affectation de cellules à des commutateurs peut être aussi perçu comme un problème de partitionnement (Merchant et Sengupta, 1994). Dans ce cas, chaque cellule i avec un volume d'appels λ_i donné est représenté par un *nœud primaire* et un certain nombre de *nœuds secondaires* (*généralement* $K.\lambda_i - 1$). K étant un facteur multiplicatif qui permet de transformer les volumes d'appels fractionnaires en nombres entiers. Les nœuds secondaires et primaire sont reliés par des arcs de coûts très élevés afin de toujours

assurer leur appartenance à une même cellule. Un arc *(i,j)* a comme coût la somme des coûts de transfert entre les cellules *i* et *j*. Les coûts de liaison sont eux proportionnels aux distances. Pour chaque paire de cellules, l'arc *(i,j)* est reporté entre les nœuds primaires des cellules *i* et *j* respectivement. Chaque commutateur *k* est à son tour représenté par un nœud et demeure associé à un sous-ensemble de taille donnée. Pour intégrer le coût de liaison entre cellules et commutateurs, on ajoute un arc de coût entre le nœud primaire de la cellule *i* et le nœud représentant le commutateur *k*. De cette manière, les méthodes de résolution du problème de partitionnement de graphes peuvent s'appliquer au problème d'affectation de cellules.

Ainsi donc, le problème d'affectation de cellules à des commutateurs s'apparente à des problèmes NP-difficiles. En particulier, il intègre une fonction de coût et des contraintes d'affectation et peut être classé comme un problème d'affectation pour lequel il n'existe que des heuristiques de résolution permettant de trouver des solutions assez proches de l'optimum.

2.4 Méthodes classiques de résolution du problème d'affectation de cellules

Nous exposerons dans ce qui suit les différentes méthodes de résolution du problème d'affectation de cellules. Ces techniques sont assez récentes et permettent d'obtenir de bons résultats. Néanmoins, elles contiennent aussi des limitations que nous développerons tout au cours de ce paragraphe.

2.4.1 Application de la méthode de Merchant et Sengupta

Pour la résolution du problème d'affectation de cellules à des commutateurs, Merchant et Sengupta (1994, 1995) sont les premiers à avoir présenté une méthode qui lui est adaptée. Dans cette heuristique, pour la domiciliation simple, les cellules sont ordonnées en nombre décroissant de leur volume d'appels. Pour chaque cellule $j = 1, \ldots, n$, on doit maintenir l'affectation au commutateur qui minimise le coût total des cellules déjà affectées selon l'algorithme suivant:

Étape 0: Trouver une affectation initiale ;

Mettre les cellules en nombre décroissant du volume d'appels. Au départ, l'affectation est vide ;

Pour chaque $j = 1, 2, \ldots, n$;

Étendre les affectations précédemment retenues en leur ajoutant toutes les affectations possibles de la cellule j à tous les différents commutateurs. Retirer toutes les affectations qui ne remplissent pas la condition sur la capacité ;

S'il ne reste aucune affectation, alors fin du programme, l'algorithme a échoué ;

S'il en reste b ou moins, les retenir toutes ;

Sinon retenir les meilleures b solutions suivant le meilleur coût total obtenu en affectant les j premières cellules ;

Retourner la meilleure des b affectations trouvées.

Le paramètre b est fixé dès le début et doit être assez grand pour permettre d'arriver à une bonne solution initiale. On introduit des mouvements dans le but de raffiner la solution initiale trouvée. Les étapes sont les suivantes:

Étape 1: Effacer les marques de toutes les cellules précédemment marquées.

Étape 2: Trouver un meilleur mouvement faisable: c'est-à-dire trouver le commutateur k auquel affecter la cellule i qui diminuerait le coût total d'une plus grande valeur ;

S'il n'y a pas d'affectation réduisant le coût, choisir celle qui augmente le moins le coût.

Étape 3: Affecter la cellule i au commutateur k, marquer la cellule i et noter le schéma d'affectation courant.

Étape 4 : S'il reste des cellules non marquées, retourner à l'Étape2.

Étape 5: Choisir le schéma d'affectation ayant la plus petite fonction objective.

Étape 6: Si la solution obtenue est inférieure à celle du schéma courant, alors la nouvelle solution devient la solution courante et on retourne à l'Étape1 ;

Sinon, on arrête.

Pour la domiciliation double, les mêmes auteurs Merchant et Sengupta (1995) ont aussi proposé un algorithme de résolution. Ils considèrent le problème de domiciliation double comme une superposition de deux problèmes de domiciliation simple. On recherche donc la solution à un premier problème de domiciliation simple correspondant à un des patrons de la journée. La solution finale obtenue est considérée comme solution initiale au second problème de domiciliation simple, c'est-à-dire le deuxième patron. Toutefois, on ne doit pas prendre en compte le coût de liaison d'une cellule lorsque celle-ci se retrouve être affectée au même commutateur que dans le premier patron. On itère ainsi d'un patron à l'autre jusqu'à l'obtention d'une solution qui ne peut plus être améliorée. Celle-ci correspondant à la solution au problème de domiciliation double. Les étapes de l'algorithme pour la domiciliation double sont:

Étape 0: Former deux problèmes d'affectation avec domiciliation simple correspondant aux différents patrons de la journée.

Étape 1: Résoudre les deux problèmes ainsi définis suivant l'algorithme de domiciliation simple ;

Soient Q le problème dont la solution A est la plus petite des deux solutions ainsi obtenues et Q' le second problème.

Étape 2: Si Q_1 désigne un problème d'affectation avec domiciliation simple identique à Q' et d'un coût de liaison nul pour toute affectation de la cellule i à un même commutateur k, que dans la solution A. Résoudre Q_1 et soit A' sa solution.

Étape 3: Même supposition pour Q avec un problème identique Q_2 où le coût de liaison est nul pour toute cellule i affectée à un même commutateur k que dans A'. Résoudre Q_2 et mettre à jour la solution A qui devient solution de Q_2.

Étape 4: Répéter les étapes 2 et 3 jusqu'à ce que les résultats des affectations ne donnent plus une amélioration de la fonction objectif. Les solutions fournies par A et A' forment la solution de la domiciliation double.

2.5 Autres heuristiques de recherche

L'heuristique de Merchant et Sengupta appliquée au problème avec domiciliation simple se compare bien à la méthode de programmation en nombres entiers pour les problèmes de petite taille. Mais elle dépend essentiellement de la solution initiale et utilise des mécanismes assez complexes pour échapper au piège du minimum local. D'autres heuristiques ont été développées pour le problème d'affectation. Nous passerons en revue les plus récents travaux effectués dans ce contexte et dont l'application au problème d'affectation fournit de bonnes solutions.

2.5.1 Heuristique de recherche taboue (RT)

La méthode de recherche taboue constitue une amélioration de l'algorithme de descente qui permet d'éviter le piège du minimum local. Elle fut introduite en optimisation combinatoire par indépendamment Glover (1986) et Hansen (1986) pour la résolution de problèmes difficiles. C'est une méthode qui part d'une solution initiale supposée locale et sur laquelle différents mouvements sont effectués pour arriver à une meilleure solution. Ainsi, pour y arriver, l'algorithme accepte de temps en temps des solutions qui n'améliorent pas toujours la solution courante. Le retour vers des solutions déjà visitées est interdit en conservant une *liste taboue T* de longueur k comportant les k dernières solutions visitées jusque là. Le choix de la prochaine solution est alors effectué sur un ensemble des solutions voisines ne comportant aucun des éléments de cette liste. Lorsque le nombre k est atteint, chaque nouvelle solution qui devient taboue remplace la plus ancienne dans la liste. L'exploration de l'espace de recherche peut être représentée par un graphe $G = (X, A)$, où X désigne l'ensemble des solutions et A l'ensemble des arcs $(x, m(x))$, $m(x)$ étant la solution obtenue en appliquant le mouvement m à x. Le graphe ainsi obtenu est symétrique car, pour chaque arc $(x, m(x))$, il existe un arc $(m(x), x)$ obtenu en appliquant le mouvement inverse m^{-1} à m. La recherche taboue part donc d'une solution initiale x_0 qui est un nœud du graphe G, et cherchera dans G un chemin

$x_0, x_1, \ldots, x_l$ où $x_i = m(x_{i-1})$ avec $i = 1, \ldots, l$. Les arcs (x_i, x_{i-1}) du chemin sont choisis en résolvant le problème d'optimisation:

$$f(x_{i+1}) = \min f(x_i)$$

Adaptation de la méthode de RT au problème d'affectation de cellules

Pour cette adaptation, Houéto et Pierre (1999) partent d'une solution initiale obtenue à partir de la plus petite distance euclidienne sur le coût de liaison. Une composante de mémoire à court terme permet d'explorer le voisinage de cette solution tout en évitant les cycles. L'espace de recherche est choisi libre des contraintes de capacité sur les commutateurs mais respecte la contrainte d'affectation unique des cellules. Chaque nouvelle solution obtenue est évaluée suivant deux critères. Le premier est lié au coût calculé à partir de la fonction objectif, le deuxième prend en compte une sanction introduite pour le non respect de la contrainte de capacité. On essaie alors de choisir à chaque étape la meilleure solution suivant ces deux critères. Trois structures de mémoires permettent d'éviter des cycles autour d'un optimum local et de raffiner la recherche.

Mémoire à court terme

La mémoire à court terme permet d'améliorer la solution courante à partir des deux critères de coût et de sanction associés à chaque solution. On définit le voisinage $N(S)$ d'une solution S comme étant l'ensemble de toutes les solutions accessibles de S par l'application d'un mouvement $m(a, b)$ à S.

m(a, b) = réaffectation de la cellule a au commutateur b

Le choix d'une solution parmi l'ensemble des solutions voisines est effectué à l'aide de la fonction de gain $G_s(a, b)$ et qui est définie comme suit:

$$G_s(a,b)=\begin{cases} \sum_{i=1, i\neq a}^{n} (h_{ai}+h_{ia}).x_{ib_0} - c_{ab_0} - \sum_{i=1, i\neq a}^{n} (h_{ai}+h_{ia}).x_{ib} + c_{ab} & \text{si } b\neq b_0 \\ M & \text{sinon} \end{cases} \qquad (2.11)$$

où,

b_o désigne le commutateur de la cellule a dans la solution S, c'est-à-dire avant l'application du mouvement $m(a, b)$;

M un nombre arbitrairement élevé.

On peut exprimer M comme étant le gain obtenu en affectant la cellule a au commutateur b_O au lieu de b. À chaque itération, vu qu'on veut minimiser le coût total des affectations, on choisira le mouvement ayant le gain minimum. Le gain sera pris égal à une valeur arbitrairement grande lorsqu'on aboutit à une même affectation. Le coût de la nouvelle solution est obtenu par la sommation suivante:

$$f(S') = f(S) + G_S(a, b) \qquad (2.12)$$

Si aucune des solutions n'engendre une amélioration du coût, alors on choisit la solution dégradant le moins la solution courante. Une liste taboue d'une certaine taille permet de garder le mouvement inverse $m(a, b')$ où b' désigne le commutateur auquel la cellule a était affectée avant le mouvement $m(a, b)$. Après un nombre k_{max} d'itérations, ayant abouti à des solutions consécutives non faisables, on introduit progressivement une pénalité sur la capacité, traduite par un multiplicateur qui est incrémenté jusqu'à une valeur maximum. Ce mécanisme de rappel se trouve désactivé dès que l'on tombe sur une solution réalisable. Lorsqu'un mouvement tabou conduit à une solution dont l'évaluation est meilleure que la solution courante, alors on annule son critère tabou.

Mémoire à moyen terme:

Cette structure de mémoire permet de revenir à des régions prometteuses pour y intensifier la recherche localement. Elle complète donc la structure précédemment décrite en ce sens qu'elle permet de ramener la recherche à des zones omises ou peu exploitées par la mémoire à court terme. Si on suppose que les bonnes solutions sont proches l'une de l'autre dans le problème d'affectation de cellules, trouver les régions prometteuses peut se faire en conservant une liste *FIFO* des dernières meilleures solutions accompagnées de leurs valeurs de gains. Une fois qu'on a déterminé les régions prometteuses, on applique les mécanismes d'intensification dont le rôle est de

diriger la recherche vers de bonnes solutions jusque là non visitées. Les deux mouvements d'intensification utilisés sont:

i_1 (a, c): permutation des cellules *a* et *c* selon les plus faibles gains ;

i_2(a, b): déplacement de la cellule *a* vers le commutateur *b* en vue de rétablir les contraintes de capacité.

C'est donc un mouvement qui est appliqué uniquement aux solutions non faisables et qui permet de réduire la pénalité.

Mémoire à long terme

C'est une structure qui permet de diversifier les différentes zones à explorer. Pour la réaliser, on met à jour un tableau dont les éléments sont le nombre de fois que chaque couple *(n, m)* (avec *n* représentant les cellules et *m* les commutateurs) apparaît dans les solutions visitées. La diversification consiste ainsi à effectuer la recherche à partir de nouvelles solutions initiales. À ces dernières sont appliqués des mécanismes de mémoire à court et moyen terme. La composante de mémoire à long terme permet alors une diversification de l'exploration du domaine en choisissant un nouveau point de départ contrastant le plus possible avec les solutions déjà visitées. Ceci permet d'explorer de manière plus efficace tout le domaine de recherche.

2.5.2 Heuristique basée sur l'algorithme génétique

Le principe de base de cet algorithme est fondé sur la théorie de la reproduction de Darwin. Holland (1975) et ses étudiants sont les premiers à l'introduire en intelligence artificielle. On crée une population initiale composée de différents chromosomes. Les éléments de cette population sont par la suite altérés à l'aide d'opérateurs génétiques pour permettre une diversité au niveau des nouveaux chromosomes. On passe alors à une évaluation de cette population. Une adaptation des algorithmes génétiques, réalisée par Hedible et Pierre (2000) se caractérise par quatre paramètres: le codage des données du problème, l'espace de recherche, la fonction d'évaluation des chromosomes parents

et le hasard dans l'évolution des chromosomes de génération en génération. De manière succincte, les étapes d'un algorithme génétique sont les suivantes:

Étape 1: Générer une population initiale de taille n représentant le nombre de chromosomes. Sélectionner au hasard les gènes qui composent le chromosome pour former une première génération.

Étape 2: Évaluer chaque chromosome par sa fonction objectif.

Étape 3: Générer de nouvelles populations et appliquer divers opérateurs génétiques pour aboutir à de meilleurs chromosomes.

Adaptation de la méthode à l'affectation des cellules

On considère une représentation non binaire des chromosomes. Les chromosomes sont normalement constitués de gènes. Dans le cas du problème d'affectation, les gènes représentent les différents commutateurs auxquels sont associées les cellules. On a donc un chromosome de longueur maximum égale au nombre de cellules présent dans le réseau. Chaque cellule (gène) peut prendre une seule valeur correspondant au commutateur auquel elle est affectée.

La population initiale est composée de différents chromosomes. Le premier est obtenu de façon déterministe en affectant chaque cellule au commutateur le plus proche. Ensuite, pour assurer la diversité au sein de la population, on crée les autres éléments ou chromosomes de manière aléatoire en se basant sur la stratégie de la population sans doubles. Évidemment la taille de cette population ne doit pas dépasser m^n, où m est le nombre de commutateurs et n le nombre de cellules. Des opérateurs de croisement et de mutation sont par la suite appliqués aux différents éléments de cette population.

L'opérateur de croisement utilisé est effectué en un lieu. Comme son nom l'indique, un croisement réfère à un changement d'un gène des chromosomes de la population. Deux chromosomes-parents, choisis de manière aléatoire dans la population, sont croisés pour fournir deux chromosomes-enfants. Les chromosomes-parents constituent les éléments de la nouvelle population. Les nouveaux chromosomes-enfants sont insérés dans cette population si et seulement si la probabilité de croisement est

respectée. Si ce n'est pas le cas, alors on inverse les chromosomes-parents avant de les insérer (on applique ainsi un opérateur d'inversion). L'opérateur de mutation, quant à lui, permet de se rassurer que l'on n'a pas ignoré certains gènes lors de la génération des populations. La taille de la population finale ainsi obtenue devient égale au double de la population initiale.

Une fois que l'on a obtenu une nouvelle population, celle-ci est évaluée suivant la fonction objective. Cette fonction associe à chaque chromosome une valeur indiquant le coût de la configuration qu'il représente. Ceci permet de classer les chromosomes de cette population en ordre croissant, qui seront par la suite évalués par rapport à la contrainte sur la capacité des commutateurs. Une fois ces deux évaluations faites, on procède à la détermination du meilleur chromosome, c'est-à-dire la meilleure série d'affectation. Toutefois, cette méthode de sélection peut n'explorer qu'une partie de l'espace de recherche, vu qu'elle ne retient que les meilleurs éléments d'une population. Pour cette raison, les auteurs de cette adaptation ont proposé la méthode de roulette de casino pour la formulation d'une nouvelle population. Le nombre de générations à analyser, quant à lui, dépend d'un nombre de cycle qui est prédéfini.

2.5.3 Heuristique du recuit simulé

C'est aussi une heuristique d'optimisation qui utilise les perturbations pour échapper au piège du minimum local en acceptant de temps à autre des solutions qui détériorent la fonction objectif. À chaque étape de l'algorithme, la solution courante est comparée avec d'autres solutions de son voisinage, obtenues à l'aide de petites perturbations. Si la nouvelle solution améliore la fonction objectif, alors elle devient solution courante et on explore son voisinage. Dans le cas contraire, si on trouve une solution qui détériore la solution courante, alors elle peut être acceptée, avec une forte probabilité au début. Celle-ci va décroître au fur et à mesure qu'on évolue dans la recherche. Le fait de considérer des solutions dont l'évaluation est inférieure à la solution courante permet de ne pas s'enfermer très tôt dans un minimum local. D'un autre coté, en diminuant progressivement la probabilité d'accepter une solution qui

n'améliore pas la fonction objectif courante, on est sûr d'atteindre (ou de ne pas laisser de coté) la bonne solution, une fois que l'on se trouve dans son voisinage. L'adaptation de cette heuristique au problème d'affectation nécessite une bonne définition des paramètres tels que la détermination d'une bonne solution initiale ainsi que son voisinage, la définition d'une solution réalisable et la détermination des différents paramètres du recuit, tels que la probabilité de transition, le critère d'arrêt, etc. Ces différents paramètres intrinsèques à l'utilisation de la méthode du recuit simulé sont aléatoires et peuvent donc s'avérer très laborieux en termes de temps de calcul.

2.5.4 Heuristique basée sur les grappes

L'idée de base est de regrouper les différentes cellules en grappes (cluster) au centre desquels se trouvent des commutateurs. On suppose à cet effet que le nombre de grappes est égal au nombre de commutateurs. L'ajout des cellules aux grappes se fait suivant le meilleur des coûts de liaison et de relève, tout en vérifiant la contrainte sur la capacité des commutateurs. Les étapes de l'algorithme proposées par Saha et al. (2000) sont les suivantes:

Étape 0: Affectation initiale

Soit Set_k l'ensemble des cellules affectées au commutateur k de capacité M_k ;

Soit c_j la cellule où se trouve le commutateur k ;

Poser $Set_k^0 = \{c_j\}$ et $M_k^0 = M_k - \lambda_j$.

Étape l (l >0):

l.1) Identifier toutes les cellules adjacentes à Set_k^l et les placer dans l'ordre décroissant de Δ = coût de relève + coût liaison ;

Choisir la cellule ayant la plus grande valeur de Δ et l'affecter à k ;

S'il en existe plus d'une, choisir celle ayant le plus de relèves avec la cellule hébergeant le commutateur. Répéter l'opération pour tous les commutateurs du réseau.

l.2) Si une cellule est adjacente à plus d'un commutateur, alors choisir l'affectation qui donne le plus petit coût de câblage entre c_j et k ;

Si la capacité du commutateur est déjà atteinte, alors choisir le second commutateur le plus proche sinon laisser la cellule de côté.

l.3) Pour chacune des affectations de c_j à un commutateur, réduire la capacité de ce commutateur du volume d'appels de cette cellule.:

Étape Finale: Si toutes les cellules ont été affectées alors l'algorithme a trouvé la solution ;

Sinon, c'est un échec.

La plupart des méthodes mentionnées dans ce chapitre utilisent des mécanismes complexes pour aboutir à une bonne solution, considérée comme la meilleure. Certaines d'entre elles, comme l'algorithme génétique, sont basées sur des mécanismes fondés sur le hasard. D'autres comme la recherche taboue se servent de mécanismes très complexes pour contrôler la recherche. Elles n'exploitent donc pas suffisamment la structure des contraintes, inhérente à ce genre de problème. La programmation par contraintes qui fait l'objet du prochain chapitre offre des mécanismes plus naturels et adaptés au problème d'affectation.

CHAPITRE 3
APERCU DE LA PROGRAMMATION PAR CONTRAINTES

La programmation par contraintes *(PC)* se définit comme l'étude des systèmes de calcul basés sur des contraintes. Elle est complémentaire d'autres techniques dérivées des algorithmes d'unification de la programmation logique et des techniques de résolution de la recherche opérationnelle. Ces dernières années, les méthodes développées selon ce paradigme ont été exploitées dans de nombreux logiciels pour la résolution de problèmes complexes relevant pour la plupart de l'optimisation combinatoire. Dans ce chapitre, nous exposerons brièvement les fondements de la programmation par contraintes ainsi que les propriétés mathématiques qui en sont à la base. Nous aborderons par la suite la description des différentes étapes et mécanismes de son fonctionnement dans la résolution des problèmes. Enfin, nous illustrerons son fonctionnement à travers quelques-unes de ses réalisations actuelles.

3.1 Évolution et concepts de base de la programmation par contraintes

L'introduction de la programmation par contraintes comme technique de résolution remonte aux années soixante avec le système *Skechpad* développé par Sutherland (1963) considéré de nos jours comme étant l'un des pionniers dans ce domaine. Par la suite, ces méthodes ont servi de point de départ pour développer plusieurs langages de programmation basés sur les contraintes et qui se sont montrés d'une certaine efficacité pour la maîtrise de problèmes complexes comme celui de planification de tâches, de transport, d'allocation de ressources, etc. C'est actuellement un domaine de recherche d'un grand intérêt. Dans ce qui suit, nous présenterons une brève évolution de la programmation par contraintes et les différents concepts essentiels à sa compréhension.

3.1.1 Évolution du langage

La programmation par contraintes est une discipline qui combine deux paradigmes déclaratifs: la résolution des contraintes et la programmation logique. Le premier système Skechpad est un éditeur graphique basé sur les contraintes. Celles-ci sont utilisées pour modéliser les relations entre les différents objets en utilisant des techniques de relaxation et de propagation de degré de liberté. Un peu plus tard ont vu le jour plusieurs autres langages parmi lesquels *Thinglab* de Borning (1981). Celui-ci utilise une interface graphique pour exprimer les contraintes servant à modéliser les comportements du système. Toutefois, les premiers environnements de programmation par contraintes n'apparurent réellement qu'après 1985. En effet, trois études ayant pour support la programmation logique furent menées parallèlement dans différents centres de recherche. *ECLIPSE* est développé à Munich par une équipe du Centre Européen de Recherche en Informatique Industrielle, *CLP($\Re$)* est étudié par un groupe de chercheurs dirigé par Watson tandis que *PROLOG III* est développé à Marseille par Colmerauer et son équipe (1987). L'activité de programmation dans ces trois systèmes passe par un mariage des contraintes avec les propriétés déclaratives de la programmation logique. Ainsi, le programmeur spécifie quoi faire et non comment le faire. La programmation logique par contraintes (*CLP*) utilise donc les concepts de la programmation logique dans lesquels la notion d'algorithme d'unification est généralisée par celle de satisfaction de contraintes.

3.1.2 Concepts de base

Comme son nom l'indique, la programmation par contraintes est une technique qui est basée sur les contraintes et les algorithmes de recherche de solution satisfaisant ces contraintes. Dans ce contexte, les trois concepts de base essentiels à sa manipulation demeurent: les contraintes, les domaines de contraintes et la satisfaction de contraintes.

Contrainte

C'est un concept relationnel. En effet, une contrainte permet d'exprimer des relations entre plusieurs variables $x_1, x_2, ..., x_n$ d'un problème. Ces dernières prennent leurs valeurs dans des ensembles $D_1, D_2, ..., D_n$. De manière formelle, on définit une contrainte comme étant une clause bâtie à partir de variables et de symboles définis dans une signature notée Σ.

Une signature comprend un ensemble de fonctions et de prédicats dont chacun est d'une certaine arité. L'arité permet de spécifier le nombre d'arguments de chaque expression.

Une contrainte primitive est une clause construite à partir de variables, de fonctions et de prédicats d'une signature Σ. Les contraintes complexes sont une conjonction de contraintes primitives. Par exemple:

> *Une contrainte définie avec des variables réelles prenant leurs valeurs dans $\Re$ peut avoir comme fonctions +, *, -, et /. Les prédicats peuvent être $=, <, \leq, >, \geq$ L'ensemble des fonctions et des prédicats forme la signature.*

Domaine de contraintes

Le domaine de contraintes *(D, L)* est constitué d'une structure *D* (le domaine de discours) et d'une classe de Σ-formules *L* (les contraintes que l'on peut exprimer). Comme exemple de domaines de contraintes, on peut citer :

- *Les contraintes booléennes*, qui permettent d'exprimer des relations entre des variables booléennes prenant des valeurs *Vrai* (représentée par 1) ou *Faux* (représentée par 0) et utilisant des opérations logiques comme la conjonction, la disjonction, l'implication. Elles sont souvent utilisées dans la modélisation des circuits logiques.
- *Les contraintes sur les arbres*, qui sont utilisées pour modéliser des structures de données utilisées en programmation, comme les listes et les arbres.
- *Les contraintes sur domaine fini*, pour lesquelles les valeurs prises par les variables appartiennent à un ensemble fini.

Le domaine constitue une notion importante en *PC*, puisqu'il détermine souvent l'algorithme de satisfaction de contraintes.

Satisfaction de contraintes

C'est la caractérisation des instanciations des variables pour lesquelles les contraintes décrites dans le problème sont satisfaites. Il s'agit de savoir si le problème défini dans son domaine admet une solution. Généralement, vu que l'on ne peut développer des méthodes de satisfaction de contrainte universelles, les algorithmes de recherche de solution sont particuliers à chaque domaine de contraintes.

L'une des méthodes utilisées pour tester la satisfaction des contraintes est de procéder par une énumération de toutes les valeurs du domaine. Ceci conduit souvent à un temps de calcul très long et n'exploite pas les contraintes définies dans la modélisation du problème. Ainsi, une deuxième méthode consiste à réécrire toutes les formules de contraintes de manière plus simple, jusqu'à l'obtention d'une expression de forme vraie. L'algorithme utilisé est semblable à celui de Gauss Jordan illustré par Mariott et Stuckey (1998) et dont les étapes sont décrites à la Figure 3.1.

À chaque étape, cet algorithme met à jour deux ensembles C et S, constitués des équations non résolues et des équations résolues respectivement. C'est un algorithme complet (c'est à dire qu'il retourne *vrai* ou *faux* à la satisfiabilité) et applicable aux contraintes linéaires sur les réels. Pour l'étendre à plusieurs autres domaines, des méthodes comme la propagation locale sont aussi utilisées, par exemple dans les algorithmes de satisfiabilité sur les contraintes arithmétiques et booléennes.

La plupart des problèmes étudiés en programmation par contraintes sont modélisés par des variables prenant leurs valeurs dans des intervalles bornés. Plusieurs algorithmes ont donc été développés par des équipes de chercheurs, pour la résolution de ces types de problèmes. Parmi les méthodes les plus couramment utilisées se trouvent les techniques de cohérence de nœuds et d'arcs, développées en intelligence artificielle, les techniques de cohérence sur les bornes introduites par la programmation par contraintes et certaines méthodes de la programmation en nombres entiers. Il convient de signaler

que les algorithmes développés sont généralement incomplets (la satisfiabilité de l'ensemble des contraintes n'est que partiellement testée). Ces algorithmes permettent néanmoins de restreindre l'espace de recherche en propageant efficacement les contraintes à chaque étape de la résolution.

Données : S est une conjonction d'équations;
C, C_0 sont des conjonctions d'équations;
c est une équation de C;
r est un nombre réel;
e est une expression arithmétique linéaire;
x une variable n'appartenant pas à e.

Initialisation: S := vrai

Tant que C non vide **faire**
C := c ∧ C_0
C := C_0
Si c peut être exprimé sans variables **alors**
Si c peut être évalué à 0 = r où r ≠ 0 **alors**
S := faux
Fin Si
Sinon
Mettre c sous la forme x = e
Remplacer x par e dans toutes les équations de C et S
S := (S ∧ (x = e))
Fin Si
Fin tant que

Figure 3. 1 Algorithme d'élimination de Gauss-Jordan

3.2 Domaine fini et résolution de problèmes combinatoires

La *PC* utilise la satisfaction de contraintes dans la recherche de solution. À chaque étape, elle essaie de vérifier si les contraintes sont satisfaites et procède par une élimination des valeurs incohérentes du domaine des variables. Le domaine fini est surtout utilisé pour modéliser les problèmes nécessitant un choix, comme ceux de transport, de planification et d'allocation de ressources, qui sont d'une grande importance dans l'industrie. Dans cette section, nous examinerons essentiellement les principes de base de satisfaction des contraintes dans les structures à domaine fini. Principalement, nous aborderons les techniques de "retour arrière", de cohérence généralisée de nœuds et d'arcs que nous illustrerons à travers certains exemples. Nous parlerons également des heuristiques de choix qui ont comme but de guider la recherche.

3.2.1 Problème de satisfaction de contraintes (CSP)

La satisfaction des contraintes sur un domaine fini désignée en Intelligence Artificielle par *CSP* (Constraint Satisfaction Problem) définit une contrainte $C = c \wedge (x_1 , D(x_1)) \wedge \ldots \wedge (x_n , D(x_n))$ *où* $V = \{x_1, x_2, \ldots, x_n\}$, est un ensemble de n variables, chacune prenant sa valeur dans un domaine $D = \{D(x_1), D(x_2), \ldots, D(x_n)\}$ et c désigne un ensemble de contraintes. Une solution d'un CSP est une affectation d'une valeur, tirée du domaine D, à chacune des variables de V de telle sorte que toutes les contraintes c soient simultanément vérifiées. En général, les algorithmes de résolution d'un *CSP* permettent juste de restreindre l'espace de recherche. Ils peuvent être décrits comme des principes de tests locaux à chaque contrainte et sont basés sur la notion de domaine réduit.

Algorithme de Retour arrière (Backtracking)

Résoudre un problème *CSP* fait partie de la classe des problèmes NP-difficiles, pour lesquels il est improbable de trouver des algorithmes polynomiaux de résolution. Le retour arrière est un algorithme complet de résolution de problème *CSP* dont le temps d'exécution est exponentiel. L'exploration du domaine est réalisée comme suit:

on part avec un choix de variable x_i de V; on examine par la suite les valeurs de D_i; si on trouve une valeur qui ne satisfait pas la contrainte, on l'élimine du domaine, et on remonte à l'étape précédente pour un autre choix (Marriott et Stuckey, 1998). Si toutes les contraintes sont respectées, alors le problème est satisfiable. Ces étapes sont montrées à la Figure 3.3. La Figure 3.2, quant à elle, illustre son application sur un exemple avec 3 variables. La taille de l'arbre de recherche obtenu dépend du choix de la variable x_i à instancier à chaque étape de l'algorithme. Certaines heuristiques de recherche sont combinées à l'algorithme pour optimiser la recherche. Néanmoins, c'est un algorithme qui demeure très coûteux en temps de calcul. Les algorithmes qui suivent sont incomplets dans ce sens qu'ils ne peuvent donner dans certains cas une solution exacte, mais ils s'exécutent dans des temps polynomiaux.

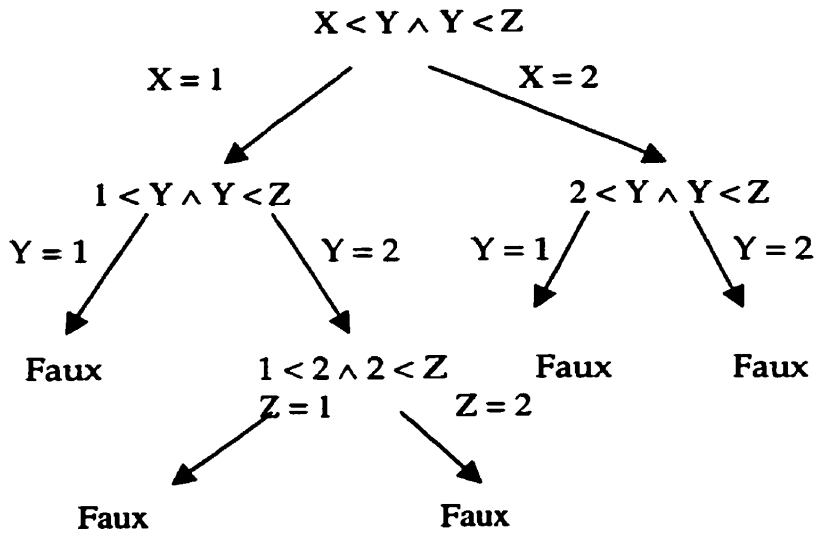

Figure 3. 2 Arbre de recherche avec retour-arrière

Exemple de retour-arrière

Soit à résoudre un problème de satisfaction de contraintes défini de la manière suivante : $(X < Y) \wedge (Y < Z)$ où X, Y, Z sont définis dans le domaine $D = \{1, 2\}$. L'exécution du retour arrière de la Figure 3.3 aboutit à une non satisfaction du CSP.

Initialisation Satisfaction := faux
Pour chaque variable $x_i \in V$ **faire**
  Tant que $D(x_i)$ est non vide
    Remplacer $x_i = e \in D(x_i)$
    Si C est satisfaisable **alors**
      Satisfaction := vrai
    Sinon
      Satisfaction := faux
      Retourner à l'étape précédente
      Retirer une autre valeur de $D(x_i)$
    Fin Si
  Fin tant que
Fin Pour

Figure 3. 3 Algorithme du retour-arrière

Techniques de cohérence de nœud et d'arc

Ces deux termes proviennent de la représentation des systèmes binaires par des graphes, dont les nœuds sont considérés comme des variables, et les arcs comme des contraintes du système. La notion de cohérence indique que la propagation est effectuée sur chacune des variables jusqu'à aboutir à une cohérence des éléments de leurs domaines par rapport aux contraintes du problème.

Technique de cohérence de nœuds

Les contraintes sont unaires:

$$vars(c) = \{x\}$$

Une contrainte primitive c est cohérente de nœuds sur un domaine $D(x)$ si, pour tout élément $d \in D(x)$, $x \Rightarrow d$ est une solution de c. Si on considère un ensemble d'équations de contraintes primitives, cette équivalence doit être vérifiée pour chacune d'elles et la satisfiabilité est testée pour toutes les valeurs du domaine $D(x)$. La Figure 3.4 illustre les étapes de l'algorithme.

Cohérence nœuds (C, D)
Soit
$C := c_1 \wedge c_2 \wedge \ldots \wedge c_n$
Pour chaque contrainte primitive c_i **faire**
Si vars(c) = 1 **alors**
$\{x\} := vars(c)$
$D(x) := \{d \in D(x) \mid (x \rightarrow d)$ est solution de c$\}$
Fin Si
Fin Pour
Retourner D

Figure 3. 4 Algorithme de cohérence de nœuds

Technique de cohérence d'arcs

Les contraintes sont binaires:

$$vars(c) = \{x, y\}$$

La cohérence d'arcs (Figure 3.5) assure la satisfiabilité du système formé des projections des contraintes basiques sur chacune des variables x. On vérifie alors si, pour

chaque valeur du domaine $D(x)$ de x, il existe une valeur du domaine $D(y)$ de y pour laquelle la contrainte est satisfaite. On élimine des domaines de x et y les valeurs qui rendent la contrainte (x, y) fausse. L'opération est effectuée pour toutes les valeurs de x et y jusqu'à l'obtention d'un domaine réduit cohérent avec les contraintes. À chaque changement du domaine de x, on doit tester de nouveau tous les arcs (z, x) avec z différent de y. Cette procédure ne garantît pas la satisfiabilité du système au complet et nécessite de ce fait une énumération des valeurs du domaine. Plusieurs autres variantes améliorées de cet algorithme ont été développées. Par exemple, la version *AC-3* utilise une procédure de propagation moins aveugle, qui évite de réviser les arcs, c'est-à-dire les contraintes non concernées par un changement du domaine de la variable x. Une autre version très utilisée est *AC-4* qui procède de la même manière que *AC-3*, mais maintient un support de liste contenant toutes les paires <*variable, valeur*>. Plutôt que d'examiner toutes les valeurs du domaine, la révision est appliquée uniquement aux valeurs de cette liste.

Technique de cohérence de bornes

Elle permet d'exprimer des contraintes comportant plus de deux variables. C'est le cas généralisé de la cohérence de nœuds et d'arcs. Pour l'illustrer, considérons le problème du sac suivant: un voyageur désire optimiser la valeur d'objets à amener avec lui en voyage. Parmi les objets qu'il doit emporter se trouvent: une bouteille de whisky de quantité 4 unités, une bouteille de parfum de quantité 2 unités et un carton de cigarettes de quantité 2 unités. Les profits tirés de chaque objet amené sont respectivement de 15, 10 et 7 dollars pour le whisky, le parfum et les cigarettes. Son sac ayant une capacité finie de 9 unités, le voyageur veut déterminer quels objets prendre pour réaliser un profit de plus de 30 dollars.

Dans ce problème, on représente chaque objet à amener par des variables. Elles prennent leurs valeurs possibles dans le domaine $D = \{0, ..., 9\}$ qui indique le nombre d'unités que le sac du voyageur peut contenir. Donc, si W représente la variable de whisky, P le parfum et C la cigarette, on pose:

$W \in \{0, \ldots, 9\}$, $P \in \{0, \ldots, 9\}$, $C \in \{0, \ldots, 9\}$.

On veut alors satisfaire les contraintes suivantes:

1. $4W + 3P + 2C \leq 9$; désigne la contrainte sur la capacité
2. $15W + 10P + 2C \geq 30$; désigne la contrainte sur le profit.

Soient

W un nouveau domaine

$C := c_1 \wedge c_2 \wedge \ldots \wedge c_n$ un ensemble de contraintes primitives

d_x, d_y des valeurs du domaine D

Faire

W :=D

Pour chaque contrainte c_i **faire**

Si $|vars(c_i)| = 2$ **alors**

Poser $\{x, y\} := vars(c_i)$

$D(x) := \{ d_x \in D(x) \mid$ pour $d_y \in D(y)$, $(x \rightarrow d_x$ et $y \rightarrow d_y)$ est solution de $c_i \}$

$D(y) := \{ d_y \in D(y) \mid$ pour $d_x \in D(x)$, $(x \rightarrow d_x$ et $y \rightarrow d_y)$ est solution de $c_i \}$

Fin Si

Fin Pour

Tant que W = D

Retourner D

Fin

Figure 3. 5 Algorithme de cohérence d'arcs

L'application de la cohérence de bornes se fait en deux étapes :

- On effectue une évaluation des bornes de chaque variable intervenant dans les contraintes 1) et 2) du problème.

$$P \leq 9/3 - 4/3\ W - 2/3\ C \Rightarrow \text{Min}_P = 9/3 - 4/3\ (\text{Max}_W) - 2/3\ (\text{Max}_C)$$

$$\text{et } \text{Max}_p = 9/3 - 4/3\ (\text{Min}_W) - 2/3\ (\text{Min}_C)$$

Les mêmes raisonnements sont aussi appliqués sur les autres variables pour trouver une formule représentant leurs bornes supérieure et inférieure.

- On calcule les nouvelles bornes des différentes variables à partir de leurs domaines et des évaluations précédentes. Chaque valeur ne satisfaisant pas ces équations est retirée du domaine. On peut aboutir ainsi à des domaines réduits de la forme :

$$P = \{0, ..., 3\};\ W = \{0, ..., 2\};\ C = \{0, ..., 4\}.$$

Il est à noter que la vérification de la cohérence dans un arbre de recherche se fait suivant divers mécanismes. Les plus connus sont les formes d'énumération implicite qui vérifient la cohérence uniquement entre les valeurs déjà assignées, le "forward checking" qui fait le contrôle de cohérence sur les valeurs fixées et celles qui ne le sont pas encore, et enfin le "Looking Ahead" qui est une forme non restreinte de cohérence permettant de vérifier la cohérence entre les valeurs non encore instanciées.

L'utilisation combinée de ces différentes techniques de propagation permet d'obtenir un algorithme complet. Comme exemple, on peut effectuer, dans un même algorithme, la cohérence sur les bornes avant le retour-arrière (backtracking) et après que les valeurs soient affectées aux variables. Dans ce cas ci, la propagation est réalisée par l'intermédiaire du domaine des variables. Les algorithmes utilisant ce principe sont considérés comme des algorithmes généralisés. Un exemple de contrainte réalisée suivant ce mécanisme est le *alldifferent* qui permet d'exprimer la contrainte globale de non-égalité sur plusieurs variables. Son algorithme est basé sur la recherche d'un couplage couvrant un ensemble de variables X, dans un graphe biparti $G(X, Y, E)$, où Y est l'ensemble des domaines de X et E l'ensemble des paires *<variable, valeur>*. Une contrainte *alldifferent* est satisfiable, si et seulement si, il existe un couplage maximum dans le graphe G correspondant. Par définition, un couplage est un ensemble d'arêtes ne partageant aucun sommet. Le couplage est dit maximum quand toutes les variables

constituent des sommets couverts par le couplage. Lorsque l'on retrouve une arête faisant partie de tous les couplages maximums, on parle d'arête vitale. L'algorithme général, implémenté dans plusieurs langages, comporte les étapes suivantes:

1. Trouver tous les couplages maximums du graphe G;
2. Identifier toutes les arêtes n'appartenant à aucun couplage maximum, c'est-à-dire ne faisant pas partie d'un chemin alterné ou d'un cycle alterné;
3. Éliminer des domaines Y, toutes les valeurs correspondantes à ces couplages.

Une autre version, dite incrémentielle et présentée à la Figure 3.6, reconsidère la contrainte *alldifferent*, suite à une réduction de certains domaines. Notons que l'application des principes de propagation sur des contraintes globales permet d'effectuer une bonne coupure de domaine. De plus, ils permettent de calculer le domaine réduit par énumération et en un temps constant. Le choix des variables à instancier ainsi que la sélection des valeurs à leur attribuer utilisent des heuristiques qui permettent de guider la recherche et de réduire la complexité de la solution.

Soit le couplage maximum courant
Reconsidérer la contrainte *alldifferent* suite à la réduction de certains domaines.
Éliminer les arêtes correspondantes
    Si l'arête est vide, **alors**
    Retourner faux
    Sinon Compléter le couplage courant
    Si il n'existe plus de couplage maximum, **alors**
    Retourner faux
    Sinon Filtrer les arêtes ne faisant partie d'aucun couplage maximum
Fin

Figure 3. 6 Algorithme de la contrainte *alldifferent* (version incrémentielle)

3.2.2 Optimisation par séparation-évaluation

Les problèmes d'optimisation sont des problèmes pour lesquels on désire trouver des solutions à un but qui minimisent une fonction de coût. La procédure d'optimisation par "séparation-évaluation" (branch & bound) s'intègre bien au modèle de résolution de la PC et peut être facilement implémentée par un solveur. Le principe de la méthode est de calculer une solution *s*, parmi l'ensemble des solutions réalisables possibles *T* du problème et de continuer la recherche en ajoutant une contrainte supplémentaire $t < s$ $(t \in T)$ sur toute nouvelle solution. Lorsque la recherche se termine en échec, le dernier coût total calculé donne le coût optimal (global) du problème. La poursuite de la recherche s'effectue en faisant un retour-arrière tout en conservant la structure de l'arbre de dérivation courant ou par itération en développant un nouvel arbre de recherche tenant compte de la contrainte supplémentaire sur la borne supérieure du coût. Tel que décrit, le "branch & bound" est une méthode dont l'efficacité dépend essentiellement de la première solution trouvée. Elle est tout indiquée dans la résolution par la *PC* de problèmes combinatoires tels les problèmes d'affectation, de planification, etc.

3.2.3 Heuristiques d'énumération

Les algorithmes de propagation permettent de réduire l'espace de recherche sur lequel des techniques d'énumération sont appliquées en vue de trouver une solution ou la meilleure solution pour un problème d'optimisation combinatoire. Ces différents algorithmes n'étant pas complets, les heuristiques d'énumération combinées avec leurs utilisations permettent de guider les choix, ce qui restreint l'espace de recherche à explorer (sans perdre la complétude du solveur). Ainsi, il existe deux grandes classes d'heuristiques qui sont exploitées. La première permet d'effectuer la sélection des variables à instancier en premier et est communément désignée par heuristique de choix de variables. La deuxième, quant à elle, permet de déterminer l'ordre dans lequel les valeurs d'instanciation d'une variable doivent être essayées.

Heuristique d'énumération de variable "échec d'abord" ("first -fail")

Le principe est de commencer la recherche par les parties du problème qui sont les plus difficiles, c'est-à-dire celles possédant le moins d'alternatives. Cette stratégie permet de se rendre compte, dès la racine de l'arbre de recherche, des choix n'ayant pas de solutions et donc aboutissant aux échecs. Son application, très variée, dépend surtout des modèles de problème. Par exemple, on peut vouloir commencer par les variables possédant les plus petits domaines ou encore raisonner suivant un impératif d'optimisation de coût en choisissant les variables possédant le moindre regret, c'est-à-dire celles possédant la plus grande valeur de différence de coût entre les deux plus bas coûts. La qualité de mesure de cette heuristique est justifiée expérimentalement et sa mise en œuvre est fonction du type d'application pour laquelle elle est utilisée.

Heuristique d'énumération de valeur "meilleur d'abord" ("best-first")

L'ordre de choix de valeurs n'affecte pas la topologie de l'arbre de recherche, mais intervient plus dans l'ordre d'exploration des différentes branches du problème. Pour cette raison, le choix des valeurs apparaît moins important dans la résolution des problèmes d'optimisation. Il peut toutefois être d'une certaine efficacité lorsqu'il s'agit de trouver une solution réalisable à un problème. Les méthodes les plus couramment utilisées sont basées sur le choix des valeurs maximum ou des valeurs les plus utilisées. Il peut s'effectuer aussi par une segmentation de tout le domaine. Le choix des valeurs est alors effectué dans chacun des sous-ensembles.

L'heuristique *"meilleur d'abord"*, très utilisée dans la recherche d'un optimum, part du principe de sélection des valeurs minimisant le terme de coût.

3.2.4 Résolution de problèmes d'optimisation combinatoire

Les caractéristiques de la PC décrites offrent des avantages tels la rapidité de la programmation, la souplesse de la mise au point et moins de risques de révision de la stratégie de résolution en cas de modification de l'énoncé. La plupart des problèmes d'optimisation combinatoire peuvent être exprimés sous forme de *CSP*. Dès lors, les langages développés en programmation par contraintes pour les problèmes sur domaine

fini peuvent être d'une certaine efficacité dans la recherche de solution de tels problèmes.

Les différentes étapes pour la résolution d'un *CSP* sont: la définition d'un ensemble de variables modélisant le problème, le choix d'une représentation des contraintes à l'aide de celles prédéfinies dans le langage utilisé et le choix des différentes stratégies de contrôle pour guider la recherche de solution dans l'espace réduit.

Définition des variables de contraintes

Elle consiste à exprimer les variables de modélisation du problème, et partant leurs domaines respectifs. Dans certains cas, ce choix peut influencer l'algorithme de résolution dans le langage choisi. Ceci s'explique par le fait que les inconnues du problème en *PC*, sont représentées en termes de variables. Résoudre le problème consiste alors à poser des contraintes sur ces variables et à déterminer les valeurs de leurs domaines qui les respectent. Débuter la recherche avec un nombre réduit de variables, donne plus de chances de réaliser la propagation des contraintes sur un espace de taille réduite. Ainsi, à cette étape de l'algorithme, plusieurs modèles sont essayés pour en retenir ceux qui aboutissent à une réduction de la taille du problème, tout en conservant l'intégrité de son formalisme.

Par exemple, pour le problème des N-reines qui consiste à placer n reines sur un échiquier $n \times n$ sans qu'elles soient en prise c'est à dire placées ni sur une même ligne, ni sur une même colonne, ni sur une même diagonale, on peut modéliser chaque reine par une variable de domaine $[1, ..., n]$ qui indique dans chaque colonne sur quelle ligne placer la reine. Ceci peut être noté: *Domaine* $([X_1,X_2, ...,X_n], [1,n])$, où X_i sont les variables.

La réduction des variables se fait en évitant certaines formes de modélisation de variables jusque là exploitées pour la modélisation en recherche opérationnelle. Par exemple, pour les problèmes d'affectation de ressources, on remplace les variables

binaires X_{ij} *(1, 0)* exprimant que i est affectée à j, par des variables S définies comme suit: $S_i = j$.

Il est aussi parfois plus efficace de considérer plusieurs jeux de variables qui correspondent à différentes vues du problème, et que l'on relie entre eux par les contraintes. Dans ce cas, il peut y avoir un coût à payer pour la gestion des relations entre les contraintes sur les différents modèles. Mais ce dernier coût peut être contrebalancé par de meilleures coupures effectuées dans chacun des modèles et par la détermination anticipée des échecs.

Définition des contraintes

Après la représentation des variables du problème, on procède à l'implémentation des contraintes sur ces variables. Généralement, on utilise les contraintes déjà définies dans le langage. Leur implémentation soigneuse permet de réduire le plus possible les domaines des variables. Le schéma de propagation d'une contrainte est le suivant:

- Réveil de la contrainte suite à une modification du domaine de l'une des variables;
- Simplification de la contrainte;
- Réduction des domaines des variables;
- Réveil des autres contraintes;
- Itération des processus de réveil et de réduction des domaines jusqu'à l'obtention de domaines stables.

Les contraintes globales, par exemple, exploitent efficacement ces procédures et permettent de réaliser des coupures du domaine à travers les différents algorithmes de propagation, coupures qui ne sont pas faisables avec les disjonctions de contraintes basiques. Comme exemple de contrainte globale, on a:

element(I, [V_1, V_2, ..., V_n], X) qui exprime que pour I=i, on doit avoir X= V_i.

I et X sont des variables du domaine, alors que V_i est une constante.

Recherche de solution

La recherche de solution est ici utilisée pour définir les étapes principales de la résolution du problème dans le langage choisi. Elle sert à relier les différentes variables aux algorithmes de sélection précédentes. Généralement, on a le choix entre les deux types suivants:

1. *Générer puis Tester*, qui procède par une utilisation passive des contraintes en évaluant d'abord les variables avant de vérifier si elles respectent les contraintes. C'est une méthode très peu utilisée car elle ne permet pas au développeur de se rendre compte assez tôt des échecs et elle peut s'avérer très dispendieuse en termes de temps d'exécution.
2. *Contraindre puis Générer*, qui se sert des contraintes définies pour réduire le domaine avant de procéder à l'évaluation. Elle procède donc par une utilisation active des contraintes, ce qui optimise la recherche par des coupures efficaces dans l'arbre de recherche.

Dans les problèmes courants, il s'agit non pas de trouver une solution mais une meilleure solution. C'est le problème d'optimisation de contraintes (*COP* – Constraint Optimisation Problem). Un *COP* est constitué d'un *CSP* et d'une fonction objectif *f*. Le but est alors de trouver une solution qui satisfait les contraintes et maximise (ou minimise) la fonction objectif. Pour cela, on ajoute une variable supplémentaire définie sur domaine fini et qui représente le coût. Tout comme en recherche opérationnelle, l'heuristique de "*Séparation et Évaluation*" (*Branch & Bound*), adaptée aux nombres entiers, est utilisée dans la recherche de solution.

3.3 Applications de la *PC* à quelques problèmes de recherche opérationnelle

Les langages de *PC* ont été appliqués à la résolution efficace de nombreux problèmes de recherche opérationnelle grâce à l'utilisation active des relations entre les contraintes. Mentionnons entre autres les problèmes de coloriage de graphe,

d'ordonnancement de tâches, de commis voyageur avec fenêtre de temps, etc. Nous présenterons dans cette section ces différentes applications.

3.3.1 Coloriage de graphe

Un problème de coloriage de graphe consiste à déterminer le nombre minimum de couleurs requises pour colorier un graphe de telle sorte que deux nœuds adjacents soient d'une couleur différente. Ce problème possède un grand intérêt en recherche opérationnelle en général pour de nombreux problèmes de conception et d'emploi de temps, et en particulier en télécommunications par exemple, pour l'allocation de fréquences dans les réseaux mobiles dans le but d'éviter les interférences entre deux cellules voisines. Fages, 1996 propose la modélisation suivante:

Modélisation et propagation des contraintes

À chaque nœud k du graphe, on associe une variable représentant la couleur et qui est définie sur un domaine $[1, ..., n]$. n est le nombre de couleurs disponibles pour colorier les nœuds du graphe.

$$\textbf{\textit{Domaine}}([N_1, N_2, ..., N_k], [1,n])$$

La contrainte d'inégalité est utilisée pour exprimer que deux nœuds i et j adjacents ont des couleurs différentes.

$$N_i \neq N_j \text{ si } i \text{ adjacent à } j$$

La recherche de solutions est basée sur l'appel d'une fonction qui utilise le "*contraindre puis générer*".

3.3.2 Affectation de fréquences dans un réseau cellulaire

Le problème de coloriage de graphes est à la base de la résolution par *ILOG Solver3.1*, du problème d'affectation de fréquences dans les réseaux cellulaires. En effet, dans les réseaux cellulaires, chaque cellule communique avec ces usagers sur des fréquences pré-allouées. Étant donné la bande de fréquence limitée disponible, on utilise une réutilisation des fréquences à l'intérieur des différentes cellules. Cette réutilisation

est rendue possible seulement dans le cas où il n'y aurait pas d'interférences entre les fréquences. Autrement dit, on exige que la distance entre les fréquences soit supérieure à une certaine valeur, au-dessus de laquelle l'utilisation d'une même fréquence par deux usagers, à l'intérieur de deux cellules, est rendue possible. Comme informations, le fournisseur dispose du nombre de cellules $(Cel_1, ..., Cel_n)$ présentes dans le réseau, du nombre de fréquences qui lui est alloué pour assurer son trafic et de la distance D entre les différentes cellules du réseau. Il s'agit alors de trouver l'affectation optimale qui minimise le nombre de fréquences en service tout en maximisant le trafic dans le réseau. On cherche ainsi à utiliser peu de fréquences pour desservir un grand nombre de cellules sans interférences, ce qui explique le parallélisme avec le problème de coloriage de graphes où on veut colorier différents nœuds avec peu de couleurs sans que deux nœuds voisins aient la même couleur.

Modélisation

1. Chaque émetteur est modélisé par une variable entière appartenant à un ensemble de n éléments représentant le nombre de fréquences disponible.
2. La contrainte sur la distance est exprimée de manière déterministe par la contrainte prédéfinie $IlcAbs(Cel_1–Cel_2) \succ D$ (ici le fait d'utiliser une contrainte symbolique au lieu de deux contraintes disjonctives $(Cel_1–Cel_2) \succ D$ *et* $(Cel_2–Cel_1) \succ D$ réduit la complexité du problème).

Sélections de variables et de valeurs

1. On commence par les variables ayant le plus petit domaine. Il est plus urgent de satisfaire dans un premier temps les émetteurs ayant moins de fréquences disponibles pour acheminer leur message et ensuite de s'occuper de ceux dont le domaine est plus large.
2. Comme choix de valeurs, on commence par les fréquences qui sont les plus utilisées. Ceci implique une mise en mémoire de chaque fréquence, chaque fois qu'une variable se trouve être bornée suite au réveil d'une contrainte.

3.3.3 Problème d'ordonnancement de tâches

Le problème d'ordonnancement de tâches est aussi désigné par «problème de planification». Dans ce problème, on dispose d'un ensemble T de tâches et d'un ensemble R de ressources. On désire assigner les différentes tâches, qui sont d'une durée déterminée, à chaque ressource. Il y a un certain nombre de tâches qui doivent être exécutées avant d'autres. Le but est d'organiser l'ordre d'exécution des tâches sur chacune des ressources de manière à minimiser le temps d'exécution, sous les contraintes suivantes:

- Deux différentes tâches ne peuvent être exécutées en même temps sur une même ressource (cas des ressources disjonctives);
- Plusieurs tâches peuvent être destinées à une même ressource, si la condition précédente est vérifiée;
- Le début de la tâche X_i appartient à un intervalle fixé. On veut ainsi débuter une tâche dans un intervalle de temps précis.

Une méthode de résolution de ce problème consiste à ordonner chaque paire de tâches partageant la même ressource, à propager les contraintes et à revenir en arrière lorsqu'un choix mène à une incohérence. La méthode présentée par Marriott. et Stuckey, 1998 est la suivante.

Modélisation

Les tâches sont représentées par des couples

$$(X_i, d_i)$$

où les variables X_i définissent le début de la tâche i et d_i sa durée. Les variables X_i sont définies dans un intervalle $[1, b]$ avec b désignant une borne supérieure représentant l'instant limite auquel la tâche i doit être débutée.

Le système de contraintes sera constitué:

- De contraintes disjonctives qui sont des inégalités linéaires de type:

$$X_i + d_i \leq X_j$$

exprimant le partage d'une même ressource par deux tâches différentes ;

- des contraintes de précédence entre tâches qui s'expriment de la façon suivante:

  Si *i* précède *j*, alors

  Prédécesseur $([X_i,d_i], [X_j, d_j])$ signifie que $X_j \geq X_i + d_i$;

- de la fonction objectif qui consiste à minimiser le temps total d'exécution des différentes tâches suivant les contraintes établies ci-dessus.

$$Minimiser(X_1, X_2, \ldots, X_n, coût)$$

Résolution du problème

Une procédure utilisée pour le choix de l'ordre d'exécution des différentes tâches sur chacune des ressources consiste à fixer une valeur d'ordre permettant l'instanciation. L'algorithme de *Séparation et Évaluation* est utilisé pour déterminer le coût optimal sur un ensemble de paires de tâches satisfaisant les contraintes de ressources.

3.3.4 Problème du commis voyageur avec fenêtre de temps

Le problème se présente comme suit (Pesant et al., 1996): Un voyageur doit effectuer une tournée dans différentes villes selon les contraintes suivantes:

1) Chaque ville doit être visitée une seule fois dans une plage horaire donnée;
2) Le voyageur dispose d'un intervalle de temps bien déterminé pour faire sa tournée qui doit se terminer au point de départ;
3) Le voyageur peut arriver dans une ville avant le temps d'arrivée prévus, à condition qu'il fasse un délai d'attente proportionnel au temps de départ prévu vers la prochaine ville;
4) On désire minimiser le temps total indispensable pour effectuer la tournée complète des villes.

Modélisation

On considère les différentes villes $V = \{1, 2, \ldots, n, n+1\}$ avec 1 et n+1 représentant la même ville départ, destination. Si on définit S_i comme une variable successeur dans le problème. On veut :

$$Minimiser \sum_i c_i S_i$$

sous les contraintes suivantes :

1) $S_i \neq S_j$, exprime que deux villes ne peuvent avoir le même successeur;
2) $S_i \neq i$, exprime qu'une variable ne peut avoir elle-même comme successeur;
3) $S_i \in \{2, \ldots, n+1\}$;
   Si β et α représentent le début et la fin d'un chemin d'une ville i, on impose les contraintes suivantes:
4) $S_i = j \Rightarrow S_{\alpha j} \neq \beta_i$, qui exclut la présence d'une même ville dans différentes chaînes;
5) $a_i \leq T_i \leq b_i$, où a_i et b_i dénotent les temps limite d'arrivée et de départ pour chaque ville i, et T_i indique la variable temps de départ;
6) $S_i = j \Rightarrow T_i + t_{ij} \leq T_j$, où t_{ij} représente la durée du parcours.

Les contraintes *5)* et *6)* définissent la fenêtre de temps pour l'arrivée dans chaque ville. Ces différentes contraintes sont propagées en *PC* pour réduire le domaine des variables. Ces contraintes participent à la réduction du domaine. La contrainte *3)* sert à initialiser le domaine des variables S_i. La contrainte *2)* permet d'éliminer les valeurs *i* du domaine des S_i très tôt pendant la recherche. La contrainte *1)* est un exemple d'utilisation du "forward-checking" *(FC)*, algorithme de propagation qui retire du domaine de la variable S_i la valeur de S_j lorsque celle-ci est instanciée. Cette procédure est valable pour l'une ou l'autre des variables S_i ou S_j. La contrainte *4)* élimine la présence de cycles dans la solution. Enfin, les contraintes *6)* et *7)* permettent respectivement d'initialiser les variables T_i et de restreindre leur domaine pendant la recherche. La complexité de la

solution est réduite par l'ajout de contraintes redondantes sur la fenêtre de temps et l'élimination de tours. Ainsi, si $t^*_{a,b}$ représente le chemin le plus court entre a et b, la contrainte:

$$(T_k + t^*_{ki} \leq T_i) \vee (T_j + t^*_{jk} \leq T_k)$$

est remplacée par:

$$(T_k + t^*_{ki} \succ T_i) \vee (T_j + t^*_{jk} \succ T_k) \Rightarrow S_i \neq j, \quad i \neq j, i \neq k \text{ et } j \neq k$$

Choix de variables et de valeurs

Le choix des variables est fait de manière dynamique. Suivant le principe "échec d'abord" qui commence par les variables ayant le domaine le plus petit. Au cas où il existerait une multitude de choix, au lieu de choisir la variable appartenant à plusieurs contraintes, on procède par élimination en retenant les variables qui peuvent aboutir à la plus petite réduction du domaine parmi ces paires. Cette nouvelle stratégie est réalisée de la manière suivante:

1. Soit s le plus petit domaine obtenu en considérant toutes les variables S_i.

   Former l'ensemble $\vartheta = \{ S_i, i = 1, ..., n : | \text{domain}(S_i) | = s \}$
2. Si s=1, alors choisir arbitrairement une variable dans ϑ;
3. Sinon:

   Pour chaque élément e appartenant à l'union des éléments du domaine des variables de ϑ, calculer le nombre d'occurrences $e^{\#}$ de e dans les domaines des variables de ϑ.

   Choisir la variable qui maximise la fonction f (v)= $\sum_{e \in \text{domain(v)}} e\#$

Le choix des valeurs est effectué suivant la plus petite durée séparant une ville de son successeur.

Dans ce chapitre, nous avons étudié les fondements de la propagation par contraintes. Plus spécialement, nous avons défini les principes de résolution des contraintes sur domaine fini, à travers leurs applications sur des exemples d'optimisation

combinatoire. Dans le chapitre suivant, nous aborderons l'adaptation de la PC au problème d'affectation de cellules à des commutateurs.

CHAPITRE 4
IMPLÉMENTATION, MISE EN OEUVRE ET RÉSULTATS

Le problème d'affectation de cellules à des commutateurs est un problème NP-difficile, faisant intervenir des variables qui prennent leurs valeurs dans un domaine fini. De ce fait, on peut le formuler sous forme de problème d'optimisation de contraintes sur domaine fini. La démarche à suivre consiste alors à exprimer les inconnues du problème à l'aide des variables de contraintes, sur lesquelles différentes règles de calcul peuvent être appliquées en vue de réduire de manière efficace leur domaine. On adopte par la suite certaines stratégies de recherche propres au problème, qui permettront d'aboutir à une bonne solution en des temps de calcul relativement acceptables. Dans ce chapitre, après une description des différentes étapes de notre adaptation, nous illustrerons le fonctionnement de celle-ci à travers une mise en œuvre sur un exemple précis. Enfin, nous présenterons les résultats obtenus et leur performance par rapport à d'autres heuristiques.

4.1 Adaptation de la PC à la résolution du problème

Le problème d'affectation de cellules à des commutateurs pouvant être formulé sous forme de *COP (FD)*, sa résolution est basée sur les différentes procédures de satisfaction des contraintes propres à ce domaine. Les trois grandes étapes à considérer dans l'algorithme sont: la modélisation du problème par le choix des variables, l'utilisation d'algorithmes de filtrage favorisant une bonne propagation des contraintes du problème sur toutes les variables et enfin la recherche efficace des solutions à travers le développement d'un ensemble de stratégies de choix de variables et de valeurs sur les domaines réduits. Notre algorithme utilise la méthode du "Contraindre et Générer". Suivant cette approche, les contraintes sont d'abord appliquées sur les variables de modélisation, pour éliminer certaines valeurs et en fixer d'autres si possible. Les solutions sont par la suite générées par énumération.

Par opposition à la méthode "Générer et Contraindre", qui fixe les valeurs aux variables avant de vérifier les contraintes, l'algorithme de programmation utilisé ("Contraindre et Générer") peut éviter que l'on considère, lors de la recherche, certaines valeurs n'aboutissant pas à des solutions faisables, ce qui permet de réduire le nombre d'échecs possibles pendant l'énumération. La méthode de "séparation et évaluation (Branch & Bound)" est utilisée pour trouver la meilleure solution. La complexité de l'algorithme au pire cas va dépendre principalement du nombre de variables modélisant le problème. Pour cette raison, contrairement à la formulation mathématique proposée par Merchant et Sengupta, 1995, et introduite au Chapitre 2, le problème est modélisé avec moins de variables, mais dont les valeurs possibles appartiennent à des domaines plus larges. Chaque cellule est ainsi représentée par une variable entière dont le domaine est déterminé par le nombre de commutateurs disponibles dans le réseau. À chacune de ces cellules sont associées deux variables de coût: le coût de câblage, qui dépend de la cellule et du commutateur auquel celle-ci est affectée, et le coût de relève qui, lui, dépend des cellules n'appartenant pas au même commutateur. Une fois le problème modélisé, les contraintes sont appliquées et propagées sur les variables choisies.

En *PC*, chaque type de contraintes possède son algorithme spécifique pour se propager. Les différents algorithmes de propagation interagissent à leur tour par l'intermédiaire du domaine des variables. Dans notre cas, nous avons deux contraintes dures à respecter. Selon la première contrainte d'unicité de l'affectation, imposée par la domiciliation simple, chaque cellule est affectée à un et un seul commutateur. Cette contrainte est respectée à coup sûr, car chaque variable entière représentant la cellule ne peut prendre qu'une seule valeur à la fois dans son domaine. La seconde contrainte concerne la capacité de chaque commutateur, qui est fixée et exprimée en volume d'appels par unité de temps. Une manière simple et efficace d'exprimer cette contrainte est de considérer chaque commutateur comme une variable ensembliste, c'est-à-dire à laquelle plusieurs valeurs peuvent être attribuées, et d'imposer que le volume total des appels provenant des cellules qui lui seront affectées ne dépasse pas la capacité dont il dispose. D'où la définition dans notre adaptation d'une autre variable ensembliste,

représentant chaque commutateur et dont le domaine des valeurs comporte l'ensemble des cellules qui lui sont affectées. Pour exprimer la contrainte sur la capacité, nous avons défini un objet permettant de déterminer et d'ajuster la capacité résiduelle d'un commutateur lorsqu'une nouvelle cellule lui est ajoutée. Si la capacité maximum est atteinte, on ne peut plus affecter de cellules à ce commutateur ou alors on revient à la valeur précédant l'ajustement de la capacité résiduelle suivant le principe du retour-arrière, et d'autres cellules dont les volumes d'appels sont transférables par le commutateur sont essayées. Dans le cas où toutes les contraintes sont satisfaites et chacune des variables est fixée, on calcule le coût total. La valeur de cette première solution trouvée est considérée comme une borne supérieure. On ajoute une contrainte supplémentaire sur la borne supérieure du coût de toutes nouvelles solutions. Dans la résolution de problèmes d'optimisation combinatoire, le coût de l'algorithme peut être prohibitif et dépend fortement du coût de la première solution trouvée. De ce fait, à chaque niveau de l'arbre, certains mécanismes de choix de variables et de valeurs sont utilisés afin de permettre de commencer la recherche par une exploration des parties du problème ayant plus de chances d'aboutir. L'un des différents avantages de la propagation par contraintes est la flexibilité d'essayer rapidement différentes stratégies de recherche sans avoir à modifier l'algorithme en profondeur. En se basant sur les statistiques du problème, on a adopté la stratégie du moindre regret sur le coût de liaison. Celle-ci est basée sur une différence entre les deux plus petites valeurs de coût de câblage possibles. L'ordre de sélection des cellules à examiner est alors fonction de la plus grande différence de coût obtenu après évaluation.

4.2 Modélisation du problème

Nous avons utilisé plusieurs jeux de variables dans la modélisation du problème. À chacune des cellules du réseau est associée une variable entière:

$$Switch_i \in \{0, ..., m-1\} \text{ pour } i = 0, 1, ..., n-1 \qquad (4.1)$$

La relation (*4.1*) permet d'initialiser le domaine des valeurs aux *m* commutateurs, pour chaque cellule. Si on considère une représentation en arbre du problème, où la racine désigne le point de départ de la recherche, et chacune des branches une alternative, résoudre le problème d'affectation consiste alors à fixer la valeur de chaque variable *Switchi*, tout en respectant la contrainte sur la capacité.

Un commutateur *j* donné est à son tour représenté par une variable ensembliste

$$Cells_j \subset \{0, 1, ..., n\text{-}1\} \text{ pour } j = 0, 1, ..., m\text{-}1 \qquad (4.2)$$

L'utilisation de plusieurs variables dans un même algorithme peut entraîner une redondance dans le modèle. Généralement, le prix à payer est la gestion des liens entre les différentes variables. Toutefois, en plus de permettre une propagation plus en profondeur des contraintes dans les différents modèles, la redondance dans la représentation est souhaitable pour combiner les avantages respectifs de facilité d'expression des variables *Switchsi* et *Cellsj*. D'un autre coté, la contrainte sur la capacité est exprimée de manière plus efficace sur les variables ensemblistes, qui se trouve être propagée sur un ensemble de cellules(réduction de la complexité).

En désignant par *cCost[i, Switchi]*, une variable de coût exprimant la valeur de la liaison entre la cellule *i* et le commutateur auquel elle est affectée, et par *hCost[i]*, une autre variable représentant le coût de relève pour une cellule, on peut formuler le problème comme suit:

$$\text{Minimiser } \sum(\text{cCost[i, Switchi]} + \text{hCost[i]}),\ i = 0, 1, ..., n\text{-}1 \qquad (4.3)$$

sous la contrainte suivante:

$$\sum_{i \in Cells_j} \lambda_i \leq M_j \qquad (4.4)$$

La relation (4.3) tend à minimiser le coût total de câblage et de relève pour toutes les cellules du réseau. La dernière relation (4.4) exprime la contrainte sur la capacité qui est ici réalisée à l'aide de l'utilisation des variables ensemblistes *Cellsj* et d'une nouvelle classe de contraintes *CapCoherence( )* que nous définirons par la suite.

Dans le langage utilisé qui est *ILOG Solver v4.4*, les variables sont exprimées de la manière suivante:

1. **IlcIntVarArray Switchs(n-1, 0, m-1)**, pour les variables entières représentant les *n-1* cellules. Leur domaine est compris entre *0 et m-1* ;
2. **IlcIntSetVarArray Cells(m-1, 0, n-1)**, utilisée pour générer les commutateurs. Une variable ensembliste est caractérisée par deux sous ensembles: celui des valeurs possibles, c'est-à-dire un sous-ensemble de toutes les cellules pouvant être affectées au commutateur qu'elle symbolise, et un sous-ensemble des valeurs requises, qui indique, à un moment donné de la recherche, toutes les cellules déjà affectées à ce commutateur. Entre ces trois ensembles se trouve la relation:

$$EnsRequis \subseteq Cells \subseteq EnsPossibles$$

   À sa création, EnsRequis=Φ et EnsPossibles = $\{0,...,n-1\}$. L'objectif est alors de trouver pour chaque commutateur le sous-ensemble des éléments requis ;
3. **IlcIntVarArray Ccost()**, est un tableau de variables, utilisé pour l'optimisation et qui en pratique permet de fixer le coût de câblage, après chaque affectation de la cellule à un commutateur ;
4. **IlcIntVarArray hCost( )**, exprime aussi un tableau de variables servant à l'optimisation et dont le rôle est de trouver un meilleur coût de relève entre les cellules affectées à des commutateurs différents;
5. **IlcRevInt CapRes**, c'est la capacité résiduelle de chaque variable ensembliste *Cellsj*. Quand une cellule est ajoutée à une de ces variables, on s'en sert pour déterminer le volume d'appels restant, et sélectionner par conséquent les cellules qui peuvent encore lui être affectée. C'est une variable réversible qui gère de manière implicite le retour-arrière en cas de violation de la contrainte sur la capacité.

4.3 Représentation des contraintes du problème

La contrainte sur l'affectation unique étant déjà respectée de par les variables modélisant le problème, nous avons défini deux nouvelles classes de contraintes: *CapCoherence( )* et *BorneInfReleve( )*. La première exprimant la contrainte sur la capacité de chaque commutateur est propagée à chaque changement du domaine des variables $Cells_j$ représentant les commutateurs. Soient *PossibleSet* et *RequiredSet*, deux ensembles d'éléments possibles et requis d'un commutateur *j*. Soient *DeltaPossibleSet* l'ensemble des éléments retirés de *PossibleSet* à une étape donnée de la recherche, et *DeltaRequiredSet*, l'ensemble des éléments ajoutés à *RequiredSet*. Lorsque la contrainte *CapCoherence( )* est appliquée au tableau de variables ensemblistes $Cells_j$, s'il y a modification du domaine d'un de ces éléments, la contrainte est appelée pour retrouver l'élément du tableau, c'est-à-dire la variable dont l'un des domaines a été changé. L'index de cette variable ainsi que le volume total des appels de toutes les nouvelles cellules ajoutées, c'est-à-dire celles appartenant à son *DeltaRequiredset*, sont utilisés pour calculer la capacité résiduelle de ce commutateur. Si la valeur trouvée est négative, alors la capacité totale du commutateur est dépassée, ce qui entraîne un échec dans la recherche de solution. Toutes les nouvelles cellules ajoutées sont retirées et on retourne à la dernière valeur calculée de la capacité résiduelle. On procède donc à une mise en mémoire de la valeur de *CapRes*, à chaque étape de la recherche. Cette opération est prise en charge dynamiquement par la bibliothèque de ILOG Solver 4.4, *IlcRev* qui utilise le retour-arrière pour revenir sur des valeurs précédentes et continuer la recherche dans une autre direction. Si au contraire la valeur de *CapRes* est positive c'est-à-dire si la capacité résiduelle du commutateur est respectée pour toutes les nouvelles cellules ajoutées, alors on parcourt toutes les cellules pour retirer de l'ensemble *PossibleSet*, les cellules qui ne sont pas encore fixées mais dont le volume d'appels ferait déborder la capacité du commutateur. La valeur de *CapRes* est par la suite mise à jour.

Pour chaque nouvelle classe de contraintes utilisée, l'algorithme comporte les trois étapes suivantes:

Étape 1: *Réveil de la contrainte*

Décide du moment d'appel de la vérification de la contrainte. Plusieurs possibilités sont offertes suivant la nature du problème. On peut le faire sur un changement de domaine, un changement des bornes de ce domaine ou enfin lorsqu'on fixe la valeur de cette variable. Pour notre cas, nous avons choisi d'effectuer le réveil de la contrainte après chaque modification du domaine de la variable $Cells_j$. Nous sommes en effet intéressés à obtenir des solutions réalisables à chaque nœud, et il parait moins restrictif de vérifier la contrainte de capacité de chaque commutateur, chaque fois que des cellules sont ajoutées ou retirées de son domaine.

Étape 2: Propagation de la contrainte

On définit la manière dont la contrainte est propagée sur les variables concernées. Ce processus permet d'éliminer des commutateurs certaines valeurs violant la contrainte de capacité.

Étape 3: *Vérification de la contrainte*

Facultative, c'est une valeur booléenne permettant d'exprimer si la contrainte est violée.

Une fois définie, cette nouvelle classe de contraintes à la Figure 4.2 est ajoutée de la manière suivante:

m.add(CapCoherence(Cells, Capacity)), *Capacity* est un tableau contenant la capacité des différents commutateurs. La valeur de *CapRes* est initialisée à celle de la capacité et est décrémentée du volume d'appels λ après chaque affectation. Un pseudo-code de l'algorithme de vérification de la condition d'ajout de chaque nouvelle cellule à un commutateur est présenté à la Figure 4.1.

La seconde classe de contraintes *BorneInfReleve( )* Figure 4.3 est utilisée pour rendre la recherche plus efficace. Elle n'est donc pas une contrainte intrinsèque au problème. Son réveil s'effectue sur un changement du domaine de la variable $Switch_i$. Au premier appel, on parcourt tous les commutateurs auxquels la cellule i peut encore

être affectée. Pour chacun de ces commutateurs, on trouve parmi toutes les cellules du réseau, celles ne pouvant plus lui être affectées et on calcule la somme du coût de relève entre la cellule *i* et ces cellules. La borne inférieure du coût de relève pour la cellule *i* est alors égale à la valeur la plus petite, obtenue sur tous les commutateurs possibles pour la cellule *i*. Cette procédure est par la suite répétée dans la propagation de la contrainte, après chaque modification du domaine. Défini de cette manière, l'algorithme utilise donc d'une manière active toutes les informations disponibles à toutes les étapes pour mettre à jour de façon dynamique la borne inférieure sur la variable de coût de relève. Les deux nouvelles classes de contrainte ainsi définies, héritent toutes de la librairie *IlcConstraintI*.

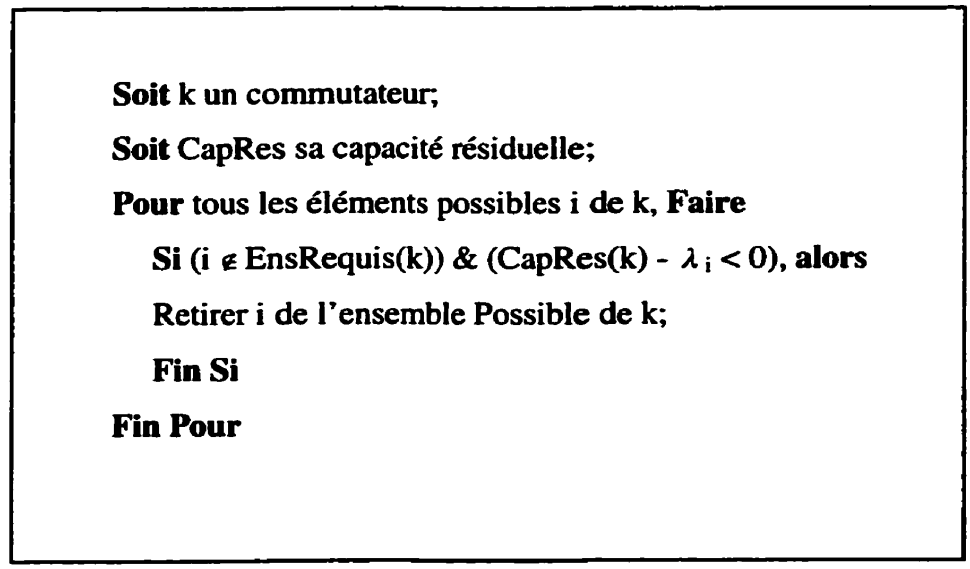

Figure 4. 1 Vérification du volume d'appels pour chaque commutateur: Exemple d'illustration du LA (Looking Ahead)

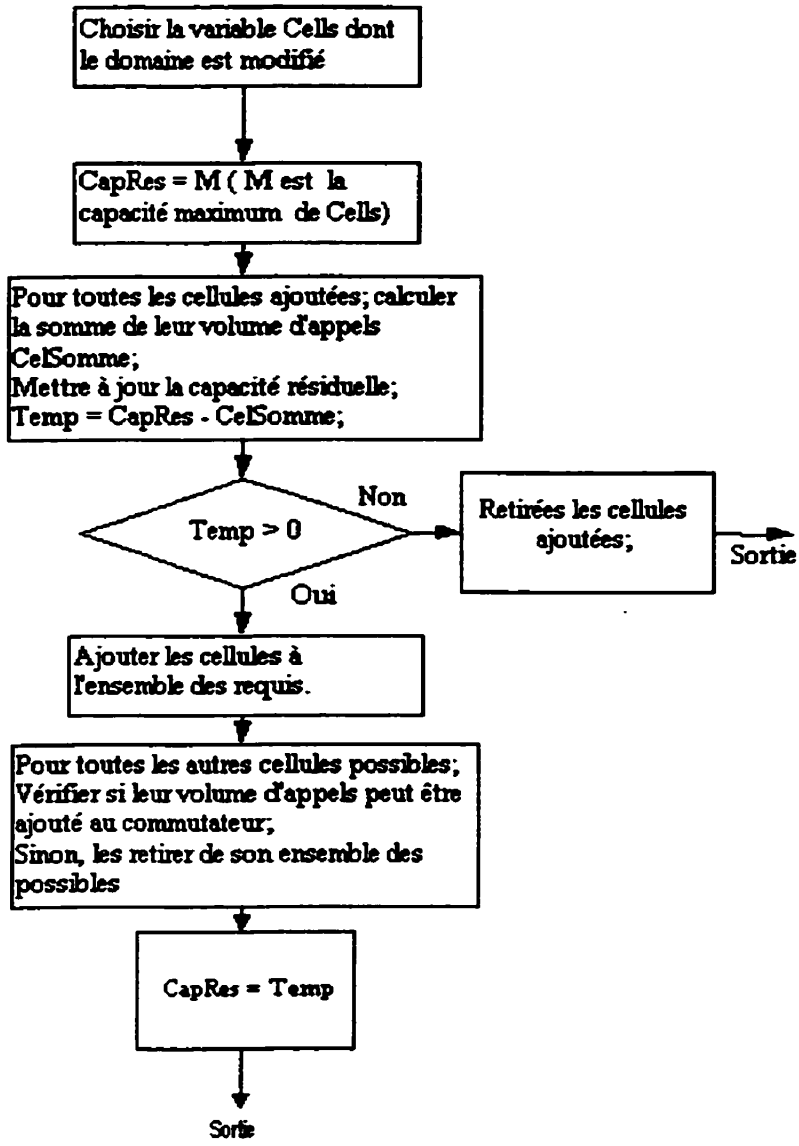

Figure 4. 2 Algorithme de la classe CapCoherence( )

Soient:

H(i) – le coût total de relève pour la cellule i;

$H(i)_{min}$ – le minimum de coût de relève pour toutes les affectations possibles;

$Switch_i$ – le commutateur auquel la cellule i est reliée;

$Cells_j$ – l'ensemble des cellules affectées au commutateur j

Pour chaque cellule i dont le domaine est modifié,
Faire $H(i)_{min}$ = Max;
    Pour chaque commutateur j, élément du domaine de $Switch_i$,
    Faire uneSom = 0;
        Pour chaque cellule i' du réseau,
        Si i' ∉ EnsPossible (Cellsj), **alors**
        uneSom = uneSom + CoutdeReleve [i][i'] (On ajoute à H(i) le coût de relève entre les cellules i et i');
        Fin Si
        Fin Pour
    Si uneSom < $H(i)_{min}$, **alors**
    $H(i)_{min}$ = uneSom;
    Fin Si;
    Fin Pour;
Mettre le minimum de H(i) à $H(i)_{min}$
Fin Pour;

Figure 4. 3 Algorithme de la classe *IlcBorneInfReleve ( )*

L'utilisation d'une variable ensembliste *Cells* pour représenter des cellules affectées à un commutateur permet d'exploiter une contrainte globale définie dans la librairie d'ILOG Solver v4.4: *IlcNullIntersection( )*. Cette contrainte est très efficace lors de la propagation étant donné qu'elle l'effectue sur plusieurs variables suivant différents algorithmes de filtrage. De manière globale, *IlcNullIntersection( )* est une contrainte qui permet d'assurer l'unicité de l'affectation en comparant les domaines de toutes les variables *Cellsj*. Pour ce faire, dès qu'une cellule est fixée à une valeur donnée, celle-ci est supprimée du domaine possible de tous les autres commutateurs. Cette opération peut paraître longue, mais elle demeure très pratique dans la propagation de la contrainte sur plus d'une variable.

La méthode de *PC* utilisée manipule deux variables principales à la fois: *Cells* et *Switch*. L'interaction, c'est-à-dire la propagation des contraintes sur une de ces variables est prise en compte par la seconde variable à travers différents jeux de démons. On peut définir un démon comme étant un ensemble d'opérations à effectuer sur une ou plusieurs variables du problème, suite à un changement de domaine, un changement des bornes de ce domaine ou alors une fixation de la valeur de ces variables, généralement causés par la propagation de l'une des contraintes. Le premier démon utilisé, *SwitchtoCellDemon,* est appelé sur un changement de domaine. Il permet de parcourir le domaine *Delta* de la variable *Switchi*. Le domaine *Delta* d'une variable entière est constitué des valeurs enlevées du domaine des valeurs possibles de cette variable et ne pouvant plus lui être attribuées. Après chaque retrait, on élimine du domaine des valeurs possibles du commutateur indiqué, la cellule i correspondante. En pratique, ceci veut dire que lorsqu'on décide pour une raison donnée qu'une cellule ne peut être reliée à un des commutateurs du réseau, alors on retire cette cellule de l'ensemble des cellules possibles de ce commutateur. Si par contre la variable est bornée, c'est-à-dire qu'on lui trouve une valeur satisfaisant les contraintes, alors on l'ajoute à l'ensemble des requis de ce commutateur. *SwitchtoCellDemon* sert donc à passer des variables entières aux variables ensemblistes. Le second démon *CellstoSwitchDemon* est lui aussi appelé sur un changement du domaine de la variable ensembliste *Cells*. Pour toutes les valeurs

ajoutées à l'ensemble *Requis* d'un commutateur, on fixe la valeur de la variable ajoutée (la cellule ajoutée) à ce commutateur.

Soit *Switchi* une variable;
Si changement du domaine de *Switchi*, **Alors**
    Pour toute valeur retirée du domaine de *Switchi*, **Faire**
    Trouver le commutateur correspondant à cette valeur,
    Retirer du domaine possible de ce commutateur la cellule i,
    Fin Pour,
    Si *Switchi* est fixée, **Faire**
    Ajouter la cellule i au domaine des requis du commutateur *Switchi*
    Fin Si
Fin Si

Figure 4. 4 Algorithme du démon *SwitchtoCellDemon*

Soit $Cells_j$ une variable ensembliste,
Tant que modification du domaine de $Cells_j$ **Faire**
    Pour toutes les valeurs ajoutées au domaine DeltaRequis **Faire**
    Fixer la valeur de toutes les cellules ajoutées au commutateur j
    Fin Pour
    Pour toutes les valeurs enlevées du domaine DeltaPossible **Faire**
    Retirer du domaine de ces cellules le commutateur j
    Fin Pour
Fin Tant que

Figure 4. 5 Algorithme du démon *CelltoSwitchDemon*

Si au contraire, c'est l'ensemble des valeurs Possibles qui est modifié, cela veut dire qu'on a diminué les valeurs possibles que peuvent prendre le commutateur et on procède directement à l'élimination de ce commutateur de l'ensemble des valeurs possibles de la variable *Switchi*. Les algorithmes de propagation des deux démons sont montrés sur les figures 4.4 et 4.5.

Une fois les contraintes testées et certaines valeurs retirées du domaine, on procède à l'énumération. Les stratégies de choix de valeurs et de variables dans l'algorithme de recherche deviennent alors très importantes. Si on considère une représentation en arbre de toutes les solutions potentielles, ces stratégies permettent de contrôler l'ordre suivant lequel on examine les différentes branches de cet arbre. Ceci a pour but de diriger la recherche vers des branches de l'arbre plus susceptibles de donner de meilleures solutions, et d'éliminer très tôt celles ne pouvant conduire à de meilleurs résultats. Généralement, on développe des stratégies de choix qui ne sont pas statiques et qui prennent en compte les données du problème et ce, à chaque étape de la recherche.

4.4 Choix des variables et des valeurs

Dans ce type de modélisation, les contraintes contribuent de manière efficace à la réduction de l'espace de recherche. Cependant, afin d'améliorer la performance de l'algorithme de "Branch & Bound" utilisé, la stratégie de choix des variables et des valeurs doit être examinée et peut s'avérer efficace suivant le type de problème. En effet, les contraintes sont propagées sur l'ensemble des variables de contraintes qui modélisent le problème. Afin d'étendre les effets de chaque propagation le plus loin possible, les variables sont choisies successivement et leurs valeurs fixées. Lorsqu'une solution est trouvée, elle sert de borne supérieure à toute nouvelle solution. Suivant le choix des variables, on peut aboutir à un processus non réalisable qui sera suivi d'un retour–arrière pour essayer d'autres valeurs ou d'autres variables lorsque le domaine de celle-ci ne contient plus de valeurs satisfaisant les contraintes. Le choix des variables à fixer peut donc permettre de réduire l'espace de recherche surtout dans la résolution de problèmes combinatoires où l'on désire non une seule solution réalisable mais la meilleure solution

réalisable possible. Généralement, on peut utiliser deux procédures de choix des variables: une sélection dynamique ou une sélection statique. Dans le problème d'affectation de cellules, comme dans tout problème d'optimisation, il apparaît plus raisonnable de commencer par les variables les plus contraintes, c'est-à-dire celles ayant le plus petit domaine et apparaissant dans plusieurs contraintes. Par exemple pour plusieurs cellules, on commencera par celles qui ne peuvent être affectées qu'à un nombre réduit de commutateurs. Trouver une affectation à ces cellules en premier lieu peut amener à se rendre compte très vite des échecs et réduire les temps d'exécution. C'est ce qu'on appelle communément le principe de l'échec d'abord (first-fail principle). Cependant, la stratégie à adopter dépend du problème que l'on résout bien que suivant le même principe du plus petit domaine. Dans notre cas, nous avons choisi le moindre regret sur le coût de câblage. Cette stratégie consiste à commencer par les variables ayant la plus grande valeur de différence entre le premier plus petit coût et le second plus petit coût de câblage. De manière concrète soit à affecter la cellule i au commutateur ayant le plus petit coût de câblage parmi tous les commutateurs auxquels la cellule i peut être affectée. Soit P le prix à payer en affectant la même cellule i à un autre commutateur ayant le second plus petit coût de câblage. On commencera par les cellules pour lesquelles P est le plus grand car elles représentent les variables pour lesquelles on aura le plus grand regret, en les affectant à un commutateur plutôt qu'à un autre.

Cette manière de procéder donne des résultats très performants lorsque les cellules ne sont pas situées à égale distance des différents commutateurs auxquels elles peuvent être affectées, et aussi parce que la fonction objective dépend surtout du coût de câblage pour un nombre élevé de cellules, ce qui est généralement le cas.

Une fois la variable choisie, on doit essayer différentes valeurs de son domaine à lui attribuer. L'ordre de sélection des valeurs ne permet pas de réduire l'espace, mais permet de guider la recherche vers des solutions avec des coûts réduits. Pour ce faire, la stratégie adoptée est basée sur la plus petite distance de coût de câblage entre cellules et

commutateurs. Chaque fois que la cellule est choisie, on commence par la relier au commutateur le plus proche dans le réseau.

Ainsi donc, si pour le choix des variables on commence par celles ayant plus de chance d'échouer lors de l'optimisation du problème, pour le choix des valeurs, on est plutôt intéressé à attribuer de bonnes valeurs, pouvant conduire à un coût proche de l'optimum.

4.5 Détails d'implémentation

Dans cette section, nous présentons les détails de l'algorithme général de notre adaptation. Nous montrons d'abord comment les données sont acquises et nous donnons un résumé des différentes classes implantées et une description des algorithmes discutés dans la section précédente.

4.5.1 Acquisition de données

Pour résoudre le problème d'affectation, certaines données doivent être fournies au programme. Ces entrées sont lues, prises en compte par le programme qui vérifie s'il existe une solution réalisable, à partir de laquelle la meilleure solution sera trouvée. Certaines de ces données, comme les nombres de cellules et de commutateurs ainsi que leurs capacités respectives sont lues directement dans la fonction principale. Les autres informations sont sauvegardées dans différents fichiers, qui sont par la suite lus à partir du programme. Les fichiers sont les suivants:

1. Le fichier appelé « fichier de coût de câblage ». C'est une matrice $n \times m$, où chaque ligne i donne les coûts de câblage de la *cellule* i avec les m commutateurs du réseau.
2. Le deuxième fichier, appelé « fichier de coût de relève », est une matrice $n \times n$ qui pour chaque cellule i donne le taux de relève avec les autres cellules du réseau.

3. Le troisième fichier, qui est le fichier de volumes d'appels, fournit les données sur le trafic, c'est-à-dire le nombre d'appels effectués par chacune des cellules du réseau par unité de temps.

4.5.2 Détails des différentes classes utilisées

Les principales étapes du programme sont:

1. Lire tous les fichiers et données du problème et initialiser les différentes variables ;
2. Poster toutes les contraintes utilisées dans l'algorithme ;
3. Quand il y a changement du domaine, appliquer le démon correspondant ;
4. Si la valeur d'une cellule est fixée, calculer le coût de liaison associée ;
5. Générer les variables *Switchi* suivant les différentes stratégies de recherche définies. Dans notre adaptation, la fonction *Generate(...)*, fixe d'abord un coût de relève pour chacune des cellules, qui est par la suite mis à jour tout au long de la recherche à l'aide d'un démon *FixHandoffCost*, que nous avons nous même définis. Les variables sont par la suite choisies suivant le moindre regret sur le coût de liaison. Différentes valeurs sont par la suite attribuées à ces variables suivant le plus petit coût de câblage. Ces étapes sont répétées tant que toutes les variables ne sont pas fixées.
6. La fonction qui minimise le coût total est réalisée par l'appel à *setObjMin(sum)*. Elle met en mémoire la dernière valeur de la variable de coût trouvée, ajoute une nouvelle contrainte sur la borne supérieure du coût total à chaque itération. Cette procédure est exécutée tant qu'il existe encore des solutions au problème. Lorsqu'il n'y en a plus, le solveur retient la dernière solution trouvée et fournit le résultat.

Les expériences ont été réalisées avec *ILOG Solver v4.4*, qui est un langage orienté objet avec une bibliothèque de classes pour les contraintes sur domaine fini. Le diagramme des principales classes de contraintes utilisées est présenté à la Figure 4.6.

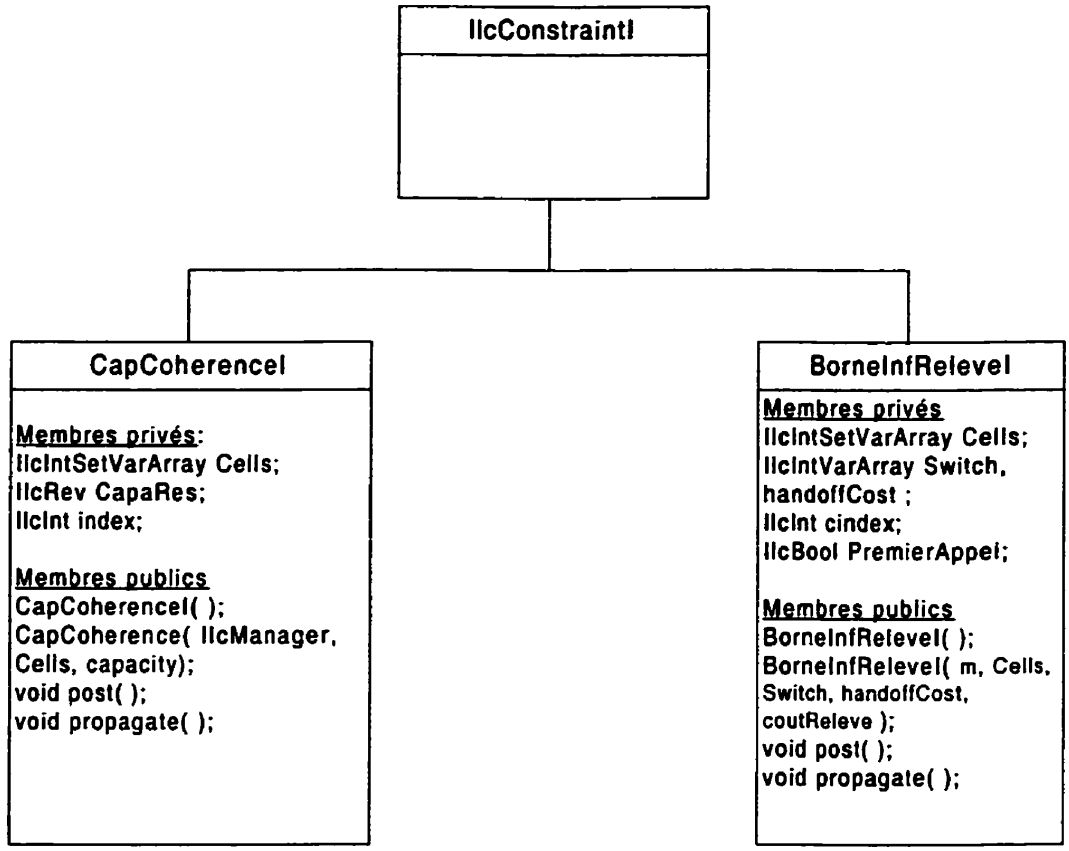

Figure 4. 6 Diagramme des principales classes de contraintes

4.6 Mise en œuvre

Pour illustrer les différentes étapes de notre adaptation de la programmation par contraintes à ce problème d'affectation, nous avons pris un exemple de fichier comportant quatorze cellules et trois commutateurs. Les données nécessaires ont été générées par un programme *Matlab* et prises de Houéto et Pierre (2000). Les cas tests

générés supposent que le coût de liaison d'une cellule à un commutateur est proportionnel à la distance qui les sépare, avec un coefficient de proportionnalité égal à l'unité. Le taux d'appel γ_i d'une cellule i est déterminé suivant une loi gamma de moyenne et de variance égales à l'unité. Les temps de séjour des appels à l'intérieur des cellules sont distribués selon une loi exponentielle de paramètre 1. Le taux de relève entre les cellules est évalué en tenant compte des cellules avoisinantes. Si par exemple une cellule possède k voisins, l'intervalle [0,1] est divisé en $k+1$ sous-intervalles en choisissant k nombres aléatoires distribués uniformément entre 0 et 1. Pour un appel qui prend fin à l'intérieur d'une cellule j donnée, on peut avoir deux issues: soit l'appel est transféré à la $i^{ème}$ cellule voisine ($i = 1, \ldots, k$) avec une probabilité de relève r_{ij} égale à la longueur du $i^{ème}$ intervalle, soit l'appel est coupé avec une probabilité égale à la longueur du $k+1^{ème}$ intervalle. Les cellules sont alors considérées comme des files d'attente M/M/1 formant un réseau de Jackson. Les taux d'arrivées α_i dans les cellules à l'équilibre sont obtenus en résolvant le système:

$$\alpha_i - \sum_j \alpha_j r_{ij} = \gamma_i \text{ avec } i = 1, \ldots, n$$

On choisit comme volume d'appel λ_i d'une cellule i, la longueur moyenne de sa file d'attente. Le taux de relève h_{ij} est définit par:

$$h_{ij} = \lambda_i . r_{ij}$$

La capacité des commutateurs est déterminée comme suit :

$$Capacité = (1+K/100)/m \sum_i \lambda_i$$

où K est choisi uniformément entre 10 et 50, ce qui permet d'obtenir une capacité de commutateur supérieure de 10 à 50% au volume d'appel des cellules et m représente le nombre de commutateurs. Les tableaux 4.1, 4.2 et 4.3 donnent respectivement les coûts de câblage entre cellules et commutateurs, de relève entre les différentes cellules et les volumes d'appels de chacune des cellules et enfin les capacités des commutateurs.

Tableau 4. 1 Coût de câblage entre cellules et commutateurs

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Comm0	0	1	1	1	1	1	1	1.73	1.73	2	2	1.73	2	1.73
Comm1	2	1.73	1	1.73	3	3	2.65	1	1	0	2	2.65	3.46	3.61
Comm2	2	1.73	2.65	3	1.73	1.73	1	2.65	3.61	3.46	2	1	0	1

Tableau 4. 2 Coût de relève entre cellules

Cellules	1	2	3	4	5	6	7	8	9	10	11	12	13
0	3	8	3	18	15	7	0	0	0	0	0	0	0
1	0	1	0	0	0	4	2	0	0	4	1	0	0
2	1	0	6	0	0	0	5	10	1	0	0	0	0
3	0	6	0	6	0	0	0	7	0	0	0	0	0
4	0	0	6	0	13	0	0	0	0	0	0	0	0
5	0	0	0	13	0	6	0	0	0	0	0	0	11
6	4	0	0	0	6	0	0	0	0	0	9	2	4
7	2	5	0	0	0	0	0	0	11	3	0	0	0
8	0	10	7	0	0	0	0	0	4	0	0	0	0
9	0	1	0	0	0	0	11	4	0	0	0	0	0
10	4	0	0	0	0	0	3	0	0	0	5	0	0
11	1	0	0	0	0	9	0	0	0	5	0	7	0
12	0	0	0	0	0	2	0	0	0	0	7	0	6
13	0	0	0	0	11	4	0	0	0	0	0	2	0

Tableau 4. 3 Volumes d'appels des cellules et capacités des commutateurs

Cellule	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Vol.appels	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Cap.comm	7	8	7											

L'algorithme essaie d'abord de trouver une solution réalisable en utilisant les principes de l'adaptation. Le Tableau 4.2 est utilisé pour sélectionner la première variable, suivant le principe du moindre regret. Par exemple, pour la cellule *zéro*, le commutateur *zéro* fournit le plus petit coût de câblage égal à 0. Si on décide plutôt d'affecter cette cellule au commutateur *deux* ou *un* qui est le second commutateur ayant le deuxième plus bas coût de liaison (égal à 2) dans le tableau, alors on induit une augmentation de coût qui est égale à 2. Ce calcul est effectué pour toutes les autres cellules du réseau. Les résultats obtenus sont inscrits au Tableau 4.4. Suivant ces valeurs, les variables de cellules à choisir sont celles ayant le plus grand regret c'est-à-dire les cellules 0, 9, 12. Étant donné que le regret se trouve être le même pour toutes ces cellules, alors la sélection se fera suivant l'ordre d'entrée de ces variables.

Tableau 4. 4 Calcul du regret sur le coût de liaison pour les cellules

Cellules	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Regret	2	0.73	0	0.73	1.65	0.73	0	0.73	0.73	2	0	0.73	2	0.73

Ainsi donc, la première variable à ajouter est celle correspondant à la cellule 0, premier élément du tableau $Switch_i$. Le choix de la valeur est basée sur la plus petite distance. Toujours suivant le tableau de coût de câblage, le commutateur *zéro* est le premier commutateur auquel la cellule 0 est affectée. Les contraintes sont appliquées sur toutes les variables. La contrainte d'unicité entraînera le retrait de la cellule 0 du

domaine des valeurs possibles de toutes les variables $Cells_j$, avec $j \neq 0$. La cellule 0 est alors ajoutée à l'ensemble des requis du commutateur 0, et la valeur de $Switch_0$ fixée à 0. On continue cette procédure jusqu'à ce que la capacité d'un des commutateurs soit atteint. Dans ce cas, on retire la dernière valeur de cellule affectée pour laquelle on sélectionne le prochain commutateur de plus petit coût. Lorsque toutes les contraintes sont enfin respectées et une affectation trouvée pour toutes les cellules du problème, alors la première solution est trouvée. Son coût est de 120 unités (Figure 4.7).

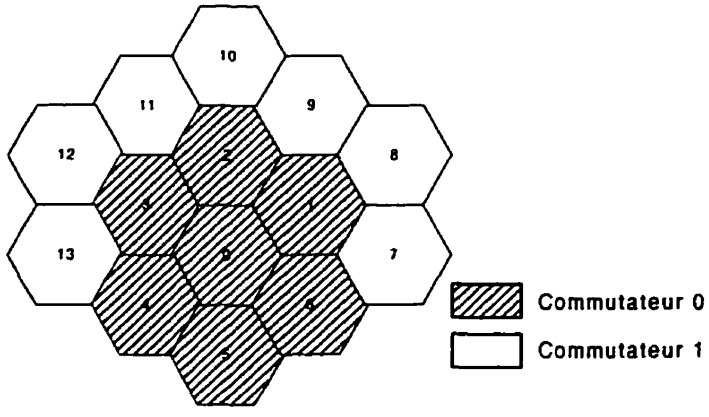

Figure 4. 7 Solution initiale de l'affectation

Le schéma d'affectation obtenu respecte la contrainte sur la capacité mais n'intègre aucun critère d'optimisation. À partir de celle-ci, on essaie de trouver des solutions meilleures en ajoutant une contrainte nouvelle sur la borne supérieure du coût total de toutes prochaines solutions suivant l'algorithme de "Séparation et Évaluation". D'autres solutions sont alors examinées en attribuant différentes valeurs aux variables générées, suivant les critères de recherche établis.

L'interaction entre la variable de coût et les autres variables permet de propager la contrainte sur les variables de modélisation, à tous les niveaux de la recherche. La

solution finale obtenue (Figure 4.8) a un coût de 75.92 unités. Dans cette affectation, aucune cellule n'est affectée au commutateur 2.

La solution obtenue respecte toutes les contraintes du problème et est trouvée en un temps de calcul raisonnable, pour ce type de réseau. Dans la prochaine section, nous allons effectuer une série de tests pour vérifier la performance de notre adaptation et aussi tester son efficacité par rapport aux autres heuristiques.

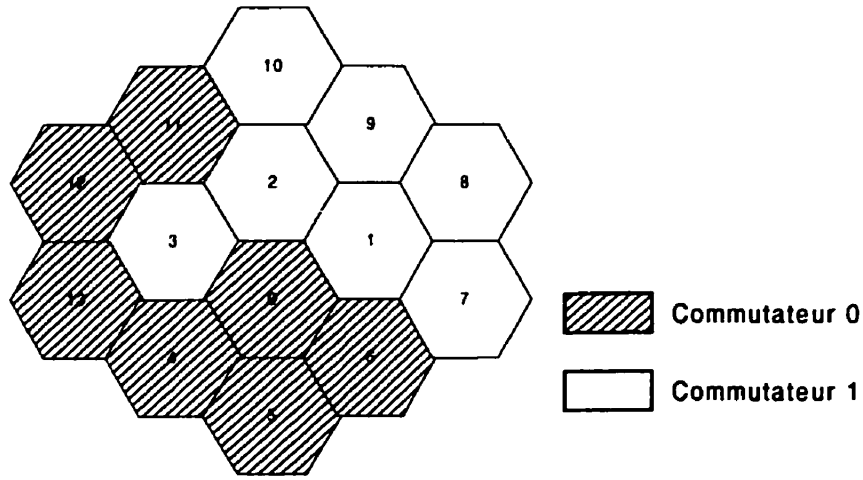

Figure 4. 8 Solution finale obtenue

$Cells_0 = \{ 0, 4, 5, 6, 11, 12, 13\}$;

$Cells_1 = \{1, 2, 3, 7, 8, 9, 10\}$;

$Cells_2 = \{\}$;

4.7 Analyse des résultats

Dans le but de mesurer la performance de l'algorithme proposé par rapport à la qualité de ces résultats, nous l'avons soumis à une série de tests. Différents cas ont été considérés avec des nombres variables de cellules et/ou de commutateurs. Dans ce qui suit, nous présenterons d'abord les résultats obtenus à partir de la mise en œuvre de

notre adaptation de l'algorithme de "*branch & bound*" (Séparation et Évaluation progressive) au problème d'affectation de cellules à des commutateurs. Nous en déduirons ainsi les relations entre la taille, le temps d'exécution, le nombre de "backtracking" (retours-arrière) et les contraintes du problème. Enfin nous ferons une comparaison de nos résultats avec ceux trouvés par d'autres approches heuristiques telles la recherche taboue, l'algorithme génétique et le recuit simulé.

4.7.1 Plan d'expériences et environnement d'exécution des tests

Les fichiers utilisés pour effectuer les différents tests ont été pris de la littérature (Houéto et Pierre, 2000). Dans un premier temps, nous avons réalisé une série de tests avec un nombre fixe de commutateurs et un nombre variable de cellules. L'objectif est non seulement d'étudier le lien entre le nombre de variables et le temps d'exécution, mais aussi d'essayer différentes stratégies de recherche pour en dégager celles qui débouchent rapidement sur de meilleurs résultats, en particulier pour les problèmes ayant une solution exacte. Pour cela, nous avons testé 20 réseaux différents pour chaque type de problème, dont le nombre de cellules varie entre 15 et 100, pour un nombre de commutateurs variant entre 2 et 4. Dans un second temps, pour dégager le comportement global de notre algorithme par rapport aux autres méthodes, nous avons réalisé différents tests sur des réseaux de petite, moyenne et grande taille. Pour les réseaux de petite taille, le nombre de cellules varie de 15 à 30 cellules, alors que le nombre de commutateurs est égal à 2 ou 3. Pour les réseaux de taille moyenne, le nombre de cellules varie de 50 à 100, pour un nombre de commutateurs compris entre 4 et 5, enfin pour les réseaux de grande taille, le nombre de cellules varie entre 150 et 200, pour un nombre de commutateurs variant entre 6 et 7. Toutes les expériences ont été réalisées sur une machine UltraSparc10. Compte tenu du fait que le temps d'exécution peut être élevé pour certains types de réseau, nous avons fixé une limite d'une journée d'exécution pour chaque type de problème.

4.7.2 Effet du nombre de cellules sur le nombre de retour-arrière

Nous avons voulu établir le rapport entre le nombre de cellules, le nombre de contraintes et le temps d'exécution. Pour effectuer cette analyse nous avons utilisé les mêmes stratégies de recherche développées à la section 4.4, c'est-à-dire celles qui sont basées sur les critères suivants:

- Choix des variables par le moindre regret sur le coût de câblage;
- Choix de valeurs suivant le plus petit coût de câblage entre cellules et commutateurs.

Suivant les résultats obtenus et qui sont présentés au Tableau 4.5, le nombre des variables de contraintes croît avec le nombre des cellules du réseau. La Figure 4.9 montre que cette croissance suit une courbe linéaire pour un nombre fixe de commutateurs. Cette remarque demeure valable pour le nombre de contraintes examinées dans les différents problèmes analysés, et ce pour un nombre fixe de commutateurs. En ce qui concerne le nombre de backtracking, les problèmes analysés ne permettent pas de déduire un comportement global de leur variation en fonction du nombre de cellules et de commutateurs dans le réseau. On a donc procédé à une analyse sur une moyenne des différents fichiers pour chaque type de problème. Les résultats obtenus sont présentés au Tableau 4.6. Celui-ci confirme le fait que l'augmentation du nombre de contraintes dans notre cas, n'entraîne pas systématiquement une augmentation du nombre de backtracking. Cette conclusion était prévisible puisque l'utilisation des contraintes non intrinsèques au problème, la modélisation par deux variables combinées avec les différentes stratégies de recherche permet d'effectuer de grandes coupures dans l'arbre. La coupure ainsi réalisée évite l'énumération avec des valeurs non optimistes qui généralement dégradent la qualité de la solution. Une analyse plus approfondie des coûts de câblage et de relève révèle toutefois que si la valeur de moindre regret, utilisée comme stratégie de recherche pour effectuer le choix des variables, est la même pour plusieurs cellules, alors on rencontre plus d'échecs avant d'aboutir à la solution finale. Par conséquent, ce choix serait inadéquat pour de tels réseaux. Enfin, pour tous les types de problèmes résolus, le temps d'exécution ne varie

pas en fonction de la taille du réseau. En effet, il est plutôt fonction du nombre de backtracking effectué, qui lui non plus n'est pas fonction de la topologie du réseau pour les raisons que nous venons d'énumérer.

Tableau 4. 5 Rapport entre le nombre de cellules et le temps d'exécution

#cellules	#comm	#var	#contraintes	#d'échecs	Tps CPU(s)
15	3	70	22	0	0.08
20	3	90	27	420	0.66
30	3	130	37	155	0.44
40	3	170	47	440	1.26
50	3	210	57	1701	4.59
60	3	250	67	170	2.16

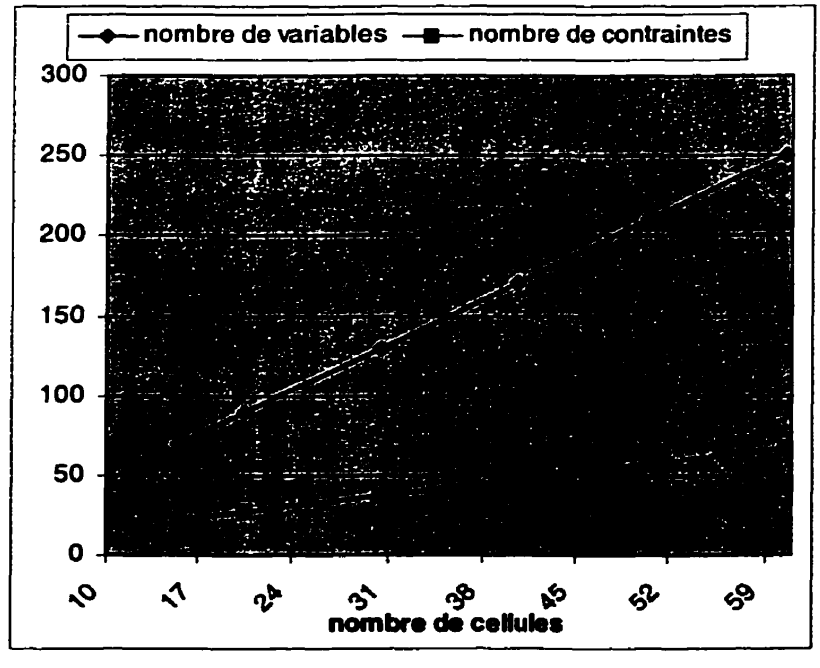

Figure 4. 9 Variation des variables et des contraintes en fonction du nombre de cellules (m=3)

Tableau 4. 6 Rapport entre le nombre de cellules et le temps d'exécution (moyenne sur un ensemble de problèmes avec m=3)

#cellules	#comm	#var	#contraintes	#d'échecs	TpsCPU(s)
15	3	70	22	400	0.25
20	3	90	27	572	1.37
30	3	130	37	8	0.33
40	3	170	47	228	0.69
50	3	210	57	4570	14.02
60	3	250	67	60745	249.12

4.6.3 Effet des contraintes sur la recherche de solution

Deux classes principales de contraintes ont été utilisées: *IlcAllNullIntersection( )* et *IlcBorneInfReleve( ).*

Contrainte *IlcAllNullIntersection( )*

La classe *IlcAllNullIntersection( )* est une contrainte globale qui permet d'exprimer la contrainte d'unicité de chaque affectation de cellules à des commutateurs, représentés dans notre adaptation par des ensembles. L'application de cette contrainte nécessite deux paramètres qui sont le tableau des ensembles sur lesquels elle s'applique et la méthode de filtrage. Dans ILOG Solver v4.4, il existe trois types d'algorithme de filtrage. Nous avons IlcLow, IlcBasic, IlcExtended. Le premier effectue le filtrage en comparant les ensembles du tableau deux à deux. Les deux autres propagent sur tous les commutateurs et s'avèrent plus efficaces en terme de nombre de variables sur lesquels ils sont propagés. Ils sont donc d'une plus grande complexité par rapport au premier algorithme de filtrage, vu le nombre d'opérations à effectuer. Afin d'examiner le gain obtenu au niveau du temps d'exécution et partant du nombre d'échecs obtenu avec l'utilisation de différents niveaux de filtrage, nous avons effectué plusieurs tests prenant

en compte ces différents algorithmes. Les résultats sont représentés au Tableau 4.7. On remarque ainsi qu'en général, quel que soit le type de filtrage utilisé, on aboutit toujours au même résultat. Le nombre d'échecs rencontrés dans chacun des différents cas est aussi le même. Cette dernière remarque indiquerait alors que les problèmes résolus ne sont pas très sensibles au type de filtrage permettant d'imposer la contrainte d'unicité de l'affectation sur les variables ensemblistes *Cells*. Ceci est du au fait que la contrainte d'unicité est trivialement respectée par les variables *Switchi*, considérées comme deuxième variable de modélisation dans notre adaptation. L'utilisation de la contrainte *IlcAllNullIntersection* n'est donc pas nécessaire pour la résolution du problème.

Tableau 4. 7 Effet de la contrainte IlcAllNullIntersection sur le nombre de retour-arrière et le temps d'exécution

#cellules	#comm	IlcLow #échecs/Tps(s)	IlcBasic #échecs/Tps	IlcExtended #échecs/Tps
15	3	0/0.07	0/0.09	0/0.08
20	3	420/0.66	420/0.68	420/0.68
30	3	155/0.42	155/0.44	155/0.44
40	3	440/1.26	440/1.24	440/1.26
50	3	1701/4.52	1701/4.52	1701/4.59
60	3	170/2.16	170/2.14	170/2.16

Contrainte *IlcBorneInfReleve( )*

La définition d'une borne inférieure sur le coût total de relève a induit une grande performance au niveau de la qualité des résultats obtenus. Comme le confirme les résultats du Tableau 4.8, pour un même réseau, le nombre d'échecs rencontrés avant d'aboutir à la solution finale est plus élevé sans cette contrainte. Par exemple, pour un réseau de 40 cellules et de 3 commutateurs, l'exécution effectue 440 échecs, en un temps

d'environ 1 seconde 26 avec l'ajout de la contrainte sur la borne inférieure, alors qu'il effectue près de 3239 échecs en 8 secondes 57 sans l'utilisation de cette contrainte. Pour certaines tailles de réseau, comme celui de 50 cellules et de 3 commutateurs, l'algorithme n'aboutit pas à une solution dans les délais indiqués. La contrainte *IlcBorneInfReleve*, réveillée à chaque modification du domaine des commutateurs et propagée de manière dynamique sur les variables fixées ou non par la méthode du LA (Looking Ahead) permet de réaliser de grandes coupures et détermine en partie l'efficacité de la méthode.

Tableau 4. 8 Effet de la contrainte IlcBorneInfReleve() sur le temps d'exécution

#cellules	#comm	Borne Inf #échecs/Tps(s)	Sans BorneInf #échecs/Tps(s)	Solution optimale
15	3	0/0.07	0/0.04	109
20	3	420/0.66	5957/6.75	195
30	3	155/0.42	352/0.81	366
40	3	440/1.26	3239/8.57	461
50	3	1701/4.52	???	588
60	3	170/2.16	170/2.52	714

4.6.4 Comparaison avec d'autres méthodes heuristiques

Le problème d'affectation de cellules à des commutateurs a été résolu avec diverses méthodes heuristiques qui sont: les algorithmes génétiques *(AG)*, la recherche taboue *(RT)* et le recuit simulé *(SA)*. Nous avons donc confronté les solutions obtenues par notre adaptation avec celles des autres heuristiques afin d'en dégager son efficacité Si nous sommes certains d'aboutir à des solutions optimales pour certains types de réseaux (jusqu'à 100 cellules), pour les grandes tailles de problèmes, la comparaison des

résultats avec les autres méthodes pourrait nous permettre de déterminer la robustesse de l'adaptation de la *PC* à ce problème d'affectation de cellules à des commutateurs. Nous nous sommes intéressés donc dans un premier temps à une étude comparative avec les algorithmes génétiques, la recherche taboue et le recuit simulé, qui sont les plus récentes méthodes appliquées à ce problème. Les tests ont été effectués sur différents réseaux soit avec un nombre variable de cellules et de commutateurs ou avec un nombre variable de cellules et un nombre fixe de commutateurs. Les résultats présentés aux tableaux 4.9 et 4.10 font état d'une performance supérieure de notre adaptation par rapport aux quatre autres heuristiques pour des réseaux avec un nombre fixe de commutateurs, vu que les solutions trouvées par l'algorithme représentent l'optimum. Toutefois, il est à remarquer que certaines méthodes comme *SA* et *AG* donnent des solutions meilleures, ce qui indiquerait probablement une réexamination des sources de leurs résultats afin de vérifier si la contrainte sur la capacité des commutateurs est respectée. Dans notre cas, toutes les solutions trouvées ont été testées et respectent cette contrainte.

Tableau 4. 9 Pourcentage d'amélioration de la PC par rapport à AG, RT, SA et HB (nombre variable de commutateurs)

#cell	#comm	PC	AG	RT	SA	HB	%PC-AG	%PC-RT	%PC-SA	%PC-HB
15	2	118	114	118	123	174	-3.50	0.00	4.06	32.18
30	3	382	394	382	405	610.	3.04	0.00	5.67	59.85
50	4	560	697	580	851	609	19.65	3.44	33.80	8.04
100	5	1389	2265	1394	1999	2078	38.67	0.35	30.51	33.15
150	6	2423	4980	2462	4271	3594	51.34	1.58	43.26	32.58
200	7	2783	3721	2768	7801	4619	25.20	-0.50	64.32	39.74

Tableau 4. 10 Pourcentage d'amélioration par rapport à AG, RT, SA et HB pour un nombre fixe de commutateurs

#cellules	#comm	PC	AG	RT	SA	HB	%PC-AG	%PC-RT	%PC-SA	%PC-HB
15	3	124	133	124	139	166	6.70	0.00	10.79	33.93
20	3	195	238	211	189	274	18.06	7.58	-3.10	40.72
30	3	364	395	364	369	487	7.80	0.00	1.35	33.95
40	3	420	424	420	611	579	0.90	0.00	31.26	37.96
50	3	588	600	588	748	813	2.00	0.00	21.39	27.67
60	3	712	917	713	832	990	22.33	0.10	14.44	39.08

Ensuite, nous avons procédé à une comparaison de notre adaptation avec l'heuristique proposée par Beaubrun, Pierre.et Conan (1999) désignée par heuristique *HB* dans la littérature, ce, pour un nombre fixe et variable de commutateurs. Les tests ont été également appliqués sur les mêmes séries de données qu'auparavant. Les résultats comparatifs avec la méthode *HB* sont aussi présentés aux tableaux 4.9 et 4.10. Une comparaison entre les méthodes développées dans la littérature et les bornes inférieures que nous avons trouvées, montre que les meilleurs résultats obtenus jusque là sont ceux fournis par la méthode de recherche taboue. Ceci s'explique par le fait que la *RT* utilise plusieurs types de mouvements, ce qui lui permet de mieux diriger la recherche et d'éviter le piège du minimum local. Le Tableau 4.11 donne une comparaison des solutions de la recherche taboue avec notre adaptation de la programmation par contraintes sur un ensemble de réseaux de taille fixe. Une comparaison entre les temps d'exécution est à l'avantage de la recherche taboue dont le critère d'arrêt est basé sur le nombre de mouvements après lequel aucune amélioration n'est obtenue. Les valeurs obtenues sont présentées aux tableaux 4.12 et 4.13.

Tableau 4. 11 Moyenne d'amélioration PC et RT (pour un nombre fixe de commutateurs)

#cellules	#comm	Sol. PC	Sol.RT	%PC-RT
15	3	118.33	122.80	3.60
20	3	178.4	186.30	4.20
30	3	335.4	336.50	0.30
40	3	452.9	452.90	0
50	3	634.2	635.11	0.10
60	3	820	821	0.08

Tableau 4. 12 Comparaison des temps d'exécution entre PC et RT (pour un nombre fixe de commutateurs)

#cellules	#comm	Temps d'exécution (sec)	
		PC	RT
15	3	0.08	0.02
20	3	10.95	0.03
30	3	30.58	0.06
40	3	0.21	0.09
50	3	24.00	0.10
60	3	0.66	0.07

Toujours dans le but d'évaluer les résultats obtenus avec notre adaptation de la programmation par contraintes par rapport à ceux de la recherche taboue, nous avons

effectué une série de tests sur un nombre de commutateurs (n = 2) et un nombre variable de cellules (m = 20 à 100). L'optimum a été trouvé pour tous les fichiers testés.

Tableau 4. 13 Comparaison des temps d'exécution entre PC et RT (nombre variable de commutateurs)

#cellules	#comm	Temps d'exécution (sec)	
		PC	RT
15	2	0.04	0.02
30	3	0.53	0.07
50	4	246.00	0.10
100	5	2615.00	0.40
150	6	3.81	0.80
200	7	6.88	0.97

À partir des tableaux 4.9 à 4.13 ainsi obtenus se dégage ce qui suit:

1. La méthode de *PC* ici développée fournit des solutions objectives optimales pour des fichiers de tests dont la taille varie entre 15 et 100 cellules pour un nombre de commutateurs égal à 2 ou à 3. En particulier, il est à noter que la solution est obtenue en des temps de calcul qui sont raisonnables.

2. Pour les réseaux de plus grande taille, les résultats que nous obtenons se trouvent aussi très satisfaisants et se comparent avantageusement aux différentes heuristiques déjà appliquées à ce problème, à part la recherche taboue. En effet, les figures 4.11 et 4.12 montrent une amélioration supérieure aux résultats fournis par les algorithmes génétiques *(AG)*, le recuit simulé *(SA)* et l'heuristique *HB*.

3. Pour la comparaison des temps d'exécution avec l'heuristique de la recherche taboue, notre méthode semble plus coûteuse. Mais le gain obtenu est assez bon pour justifier cette perte de performance. En effet les solutions que fournit l'adaptation de la *PC* se trouvent souvent proches ou parfois meilleures que celles de la *RT*, bien que ne représentant pas la solution optimale.

4. Enfin, notons que les solutions proches de celles fournies par la *RT* (lorsque celles-ci se trouvent être meilleures) pour les réseaux de grande taille sont trouvées en quelques minutes. Mais vu qu'elles ne représentent pas l'optimum, nous ne pouvons dégager le temps exact de calcul pour atteindre l'optimum global, les temps d'exécution ayant été fixés à un jour pour ces différents problèmes.

4.8 Comparaison avec une estimation de la borne inférieure

Dans la méthode de retour-arrière (branch & bound) utilisée pour notre adaptation, nous avons effectué une recherche dans tout l'espace disponible. De ce fait, toutes les solutions possibles sont explorées par l'algorithme qui, lorsqu'il se termine indique que la solution trouvée est l'optimum. Afin d'évaluer la qualité de nos solutions et aussi de dégager la distance par rapport à un optimum global pour les fichiers de grande taille dont la solution exacte n'a pu être trouvée, nous allons procéder à une comparaison avec une estimation de la borne inférieure.

4.8.1 Présentation de la méthode d'estimation d'une borne inférieure

Le problème à résoudre consiste essentiellement à trouver une affectation dont la valeur objective minimiserait le coût total de câblage et de relève, tout en respectant la contrainte sur la capacité des commutateurs. Lorsque l'on relâche la contrainte sur la capacité des commutateurs, une première borne inférieure peut s'écrire sous la forme:

$$LB1 = \sum_{i=1}^{n} \min_{k}(c_{ik}) \qquad (4.1)$$

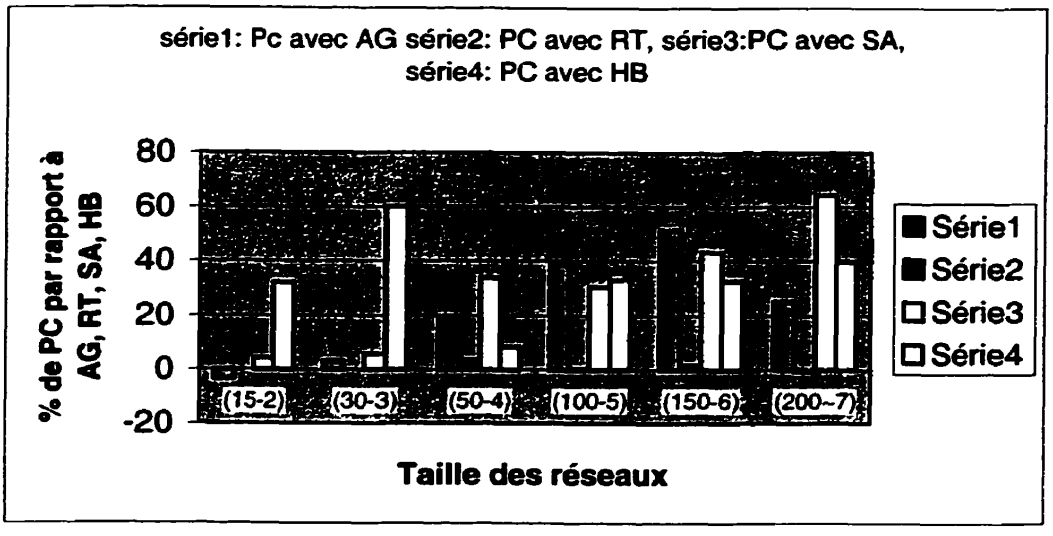

Figure 4. 10 Étude comparée pour les réseaux de taille variable (nombre de commutateurs variant)

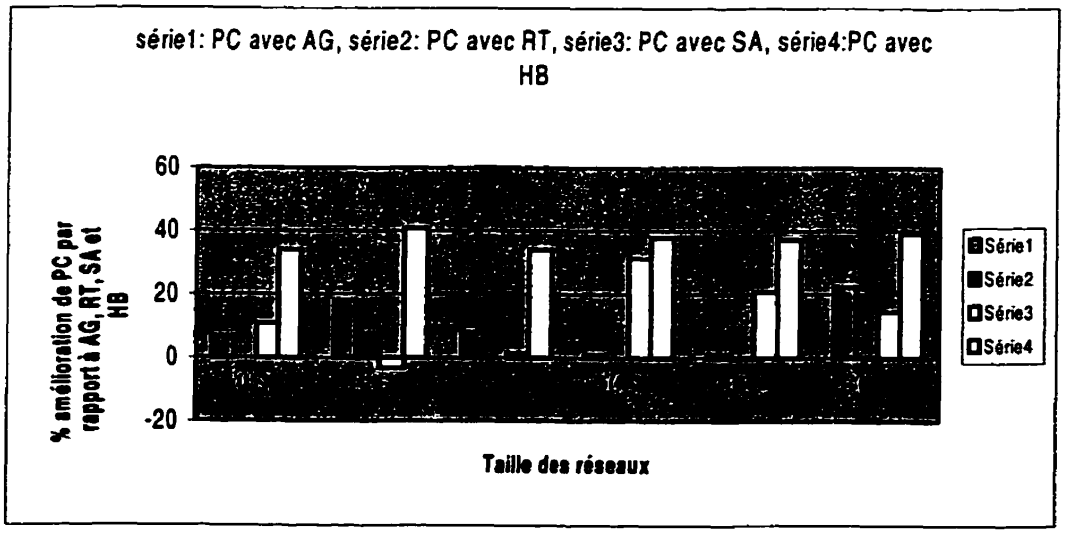

Figure 4. 11 Étude comparée pour les réseaux ayant un nombre variable de cellules (nombre de commutateurs fixe)

Malheureusement, cette borne ne tient pas compte des relèves entre cellules et commutateurs et suppose que toutes les cellules peuvent être affectées à un même commutateur. En ne considérant pas ce cas extrême, c'est-à-dire en supposant que toutes les cellules ne peuvent être affectées à un même commutateur, Houéto et Pierre, 1999 ont établi que l'on devrait aboutir à au moins une bipartition *(p,q)* de l'ensemble des n cellules du réseau. Dans ce cas, le nombre total de relèves à considérer est égal à *2pq* et on peut trouver le nombre minimal de relèves en résolvant le problème suivant :

$$\min 2pq$$

$$\text{Sujet à: } p + q = n,\ p \geq 1,\ q \geq 1 \qquad (4.2)$$

Vu que la solution de ce problème a pour bipartitions *(1,n-1)* et *(n-1,1)*, une borne inférieure pour le nombre de relèves est *2(n-1)*. Si on suppose que la matrice de coût de relève est représentée par H, et que l'on désigne par H^T sa transposée, soit h_T la matrice triangulaire supérieure de $(H+H^T)$. Une borne inférieure tenant compte des coûts de relèves devrait considérer au moins n-1 relève de h_T. Ainsi, ils ont pu trouver une nouvelle borne inférieure:

$$LB2 = \sum_{i=1}^{n} \min_{k}(c_{ik}) + \sum_{p=1}^{n-1} \sum_{q=p+1}^{n} \mathbf{1}_{N\_}\{h_T(p,q)\}.h_T(p,q) \qquad (4.3)$$

où $N\_$ désigne l'ensemble des *n-1* premiers minima de la matrice triangulaire h_T, $\mathbf{1}_{N\_}$ désigne la fonction indicatrice de l'ensemble $N\_$. avec $\mathbf{1}_{N\_}\{x\}= 1$ si $x \in N\_$ et *0* sinon. Enfin $h_T(p,q)$ désigne l'élément à la ligne p et à la colonne q de la matrice triangulaire h_T. Pour finir, notons que *LB1* $\leq$ *LB2* et donc *LB2* est a priori une meilleure borne inférieure que *LB1*.

4.8.2 Rapport entre la borne inférieure et les solutions trouvées

Avec l'architecture des réseaux de communications personnelles, une cellule possède au plus six voisines. De ce fait, lorsque le nombre de cellules augmente, la matrice H comporte plusieurs éléments nuls. Cela entraîne que les *n-1* premiers minima de la matrice triangulaire h_T sont nuls et la borne inférieure *LB2* est approximativement

égale à *LB1*. Le Tableau 4.14 montre l'écart entre les valeurs objectives de la *RT* et la borne inférieure *LB2* d'une part, et le pourcentage d'amélioration de notre adaptation par rapport à la RT pour les réseaux de moyenne taille pour lesquels nous avons pu déterminer une solution exacte. On remarque quand même un écart entre les valeurs qui s'explique par le fait que la borne inférieure *LB2* ne tient pas compte de la valeur des coûts de relève qui sont considérables pour ces types de fichiers. Les solutions obtenues pour les fichiers de plus grande taille sont présentées au Tableau 4.15. La *PC* est donc une méthode de résolution simple mais très robuste qui donne des résultats proches de l'optimum dans l'ensemble. Cette simplicité est possible au prix d'une bonne connaissance du problème à résoudre, afin d'en exploiter au maximum les relations entre les contraintes.

Tableau 4. 14 Comparaison pour m =3 de PC par rapport à RT et à la borne inférieure (BI)

#cellules	15	20	30	40	50	60
Distance(%) RT et BI	10	7	4	4	3	2
Distance(%) PC et RT	3.60	4.20	0.30	0.00	0.10	0.08

Tableau 4. 15 Comparaison de PC par rapport à RT et à la borne inférieure (BI) (pour un nombre variable de commutateurs)

Cel ~comm	15~2	30~3	50~4	100~5	150~6	200~7
Distance(%) RT et BI	10	9	9	9	9	8
Distance(%) PC et RT	0.00	0.00	3.44	0.35	1.58	-0.50

La méthode de *PC* donne ainsi des solutions exactes pour des tailles de réseaux allant jusqu'à 100 cellules. Sur d'autres, dont on n'a pas pu atteindre l'optimum, les solutions se comparent avantageusement à celles trouvées par la méthode de recherche tabou, qui fournit des coûts moindres par rapport à toutes les autres heuristiques adaptées au problème d'affectation.

CHAPITRE 5
CONCLUSION

5.1 Synthèse des travaux

Dans ce mémoire, nous avons développé et mis en œuvre une adaptation de la programmation par contraintes au problème d'affectation de cellules à des commutateurs dans les réseaux de communications personnelles. Ce problème qui consiste essentiellement à minimiser une fonction de coût composée des coût de liaisons et de relèves, tout en respectant des contraintes de capacités des commutateurs. Considérer toutes les combinaisons possibles pour en dégager la meilleure conduit très vite à une explosion combinatoire et ne peut être effectué en un temps polynomial. Ainsi pour un nombre de cellules supérieur à 15, les méthodes heuristiques jusque là développées ne fournissent pas de solutions exactes à ce problème.

Avec les récentes techniques développées en programmation par contraintes, il est possible d'utiliser de manière active les contraintes du problème pour guider la recherche et restreindre le domaine des valeurs prises par les variables de modélisation. De ce fait, en exploitant les techniques de résolution développées en recherche opérationnelle, il est possible de guider la recherche et d'espérer aboutir à de bonnes solutions en un temps raisonnable

L'algorithme utilisé est basé sur les techniques de résolution sur domaine fini. Après plusieurs expérimentations, nous avons opté pour l'utilisation de deux variables dans la modélisation du problème. En effet, les différentes contraintes imposées par le problème à résoudre sont propagées plus ou moins en profondeur suivant la variable sur laquelle elles sont appliquées. Le fait de modéliser le problème avec deux jeux de variables permet d'exploiter la performance de propagation de chacune de ces variables. L'interaction entre les variables de modélisation est à son tour réalisé grâce à l'utilisation d'une série de démons que nous avons définis et qui gèrent de manière dynamique les relations entre les variables. La recherche des bonnes solutions est

réalisée avec par le biais des stratégies qui permettent de choisir efficacement à chaque itération les meilleures variables ainsi que les valeurs à leur assigner parmi tout l'ensemble possible. La performance de l'algorithme a été améliorée grâce à la détermination d'une borne inférieure sur le coût des relèves dont l'insertion comme contrainte dans l'algorithme a permis d'aboutir très rapidement à une bonne solution initiale réalisable. Cette nouvelle classe de contraintes ajoute beaucoup de robustesse à la méthode puisqu'elle est propagée dynamiquement sur toutes les variables à chaque changement de leur domaine.

De manière générale, les solutions que nous avons obtenues sont satisfaisantes. Pour des problèmes de taille réaliste allant jusqu'à 100 cellules, nous avons pu déterminer des bornes inférieures en quelques minutes. En vue de dégager les meilleurs paramètres à utiliser, nous avons testé notre adaptation avec différents paramètres de la programmation par contraintes. Entre autres, nous avons testé l'influence des différentes stratégies de recherche, du type d'algorithme de filtrage et de l'ajout de la contrainte sur la borne inférieure du coût de relève sur la nature des solutions obtenues. Ainsi, un choix judicieux de ces paramètres a permis de retenir ceux offrant les meilleures.

Nous avons aussi effectué une comparaison de notre algorithme avec plusieurs autres algorithmes adaptés à ce problème. De cette comparaison, il résulte que la méthode proposée est la première à pouvoir fournir des solutions optimales pour des réseaux d'une certaine taille (entre 15 et 100 cellules pou 2 ou 3 commutateurs). Les heuristiques de recuit simulé, d'algorithme génétique fournissent des résultats qui sont en général très loin de l'optimum global trouvé pour ces réseaux. Quant à la recherche taboue, elle fournit des résultats très proches des valeurs de la PC. Étant donné que le temps d'exécution de l'algorithme devient considérable lorsque le nombre de cellules et de commutateurs croit, nous avons alors procédé à une analyse de nos résultats avec ceux de la RT, en fixant un temps d'arrêt d'environ une journée pour les grands réseaux. Ici aussi, les résultats obtenus sont satisfaisants et, dans certains cas, meilleurs que la RT bien que ne représentant pas l'optimum. Notons enfin la performance de notre

adaptation de la *PC*, qui branche très tôt sur de bonnes solutions lorsque la taille du réseau augmente.

5.2 Limitations des travaux

En dépit des résultats satisfaisants fournis par notre algorithme, il est important de remarquer que les solutions obtenues dépendent fortement des paramètres du problème à résoudre.

En effet le choix de la stratégie de recherche effectué dans cette adaptation est basé sur des valeurs de coûts de liaison et des relèves. Ces derniers varient fortement en fonction des données utilisées. Bien que les tests aient été effectués sur un grand nombre de fichiers, il est difficile de dire que pour n'importe quel type de fichiers nous aboutirons aux mêmes améliorations. Si on considère que dans certains types de réseaux, l'on désire principalement minimiser les coûts de relèves, les critères exploités dans cette adaptation ne fourniront pas nécessairement des solutions aussi optimistes. De plus nous avons utilisé un critère d'arrêt qui est basé sur le temps d'exécution, pour de grands réseaux. La méthode ne garantit donc pas l'obtention d'une solution exacte pour ceux-ci. Il serait ici intéressant de développer une adaptation dont le but premier serait de trouver la meilleure solution et non une solution optimale.

Notre algorithme n'utilise pas de manière efficace la redondance des contraintes exploitée en *PC* pour la réduction de l'espace de recherche. Il aurait été peut-être plus efficace d'examiner toutes les contraintes du problème pour en propager celles dont l'expression sous une autre forme peut entraîner une propagation plus en profondeur.

Notre algorithme exploite efficacement deux variables dans la modélisation, ce qui parfois augmente le nombre de variables et le temps d'exécution. Une approche possible serait peut être de considérer l'utilisation d'une seule variable, en particulier celle représentant les commutateurs et d'analyser le comportement de cette modélisation par rapport à notre adaptation.

Enfin, la *PC* n'utilise aucun critère pour contrôler la recherche à part les stratégies de recherche dont nous avions fait mention. Une fois les contraintes propagées, on

procède par énumération pour trouver la meilleure solution. Cette procédure n'utilise donc pas des mécanismes de contrôle dans le processus d'énumération.

5.3 Indication de recherches futures

Le problème d'affectation de cellules à des commutateurs pose encore plusieurs défis. La méthode de programmation par contraintes étant relativement récente, plusieurs pistes sont encore à explorer dans l'adaptation de cette méthode à la résolution dudit problème. Les futurs travaux de recherche pourraient par exemple élaborer de nouvelles stratégies dynamiques de recherche basées sur les deux composantes de coût. On pourrait par exemple élaborer une stratégie de recherche qui permettrait d'essayer les valeurs de coût de relèves lorsque les cellules se trouvent être à égale distance des commutateurs.

Il serait aussi intéressant de trouver si possible d'autres formes de modélisations plus efficaces du problème. Ceci devrait tenir compte de l'ajout de nouvelles contraintes redondantes au problème.

D'un autre côté, il serait souhaitable de tenir compte de la variation du volume d'appels à l'intérieur d'une cellule et d'inclure cette formulation dans la résolution du problème.

Et enfin, on pourrait essayer de résoudre le problème de domiciliation double avec l'approche de la programmation par contraintes. Ce dernier fait intervenir une nouvelle contrainte sur la borne inférieure, qui se trouve être différente de celle que nous avons définie dans ce mémoire compte tenu du fait que chaque cellule peut être affectée à deux commutateurs suivant les moments de la journée. Le problème à résoudre devient alors plus contraint et serait mieux résolu avec les méthodes de la *PC*.

BIBLIOGRAPHIE

BEAUBRUN R., PIERRE S. et CONAN J., An efficient method for optimizing the Assignment of Cells to MSCs in PCS Networks. *Proceedings 11th Int. Conf. on Wireless Comm. Wireless 99, Vol.1*, July 1999, Calgary (AB), pp. 259-265.

BENHAMOU F. et COLMERAUER A., Constraint logic programming: *Selected Research.* MIT Press, 1993.

BORNING A., The programming language aspects of ThingLab, a constraint oriented simulation laboratory. *ACM Transactions on programming languages and systems*, 3(4): 252-387, October 1981.

COHEN J., Constraints logic programming languages. *Communications of the ACM*, 33(7): 52-68, July 1990.

COLMARAUER A., an introduction to PROLOG-3. *Communications of the ACM*, 33(7): 69-90, July 1990.

COLMARAUER A., opening the PROLOG-3 universe. *BYTE Magazine*, 12(9), August 1987.

COLMARAUER A., PROLOG 2. Reference manual and theoretical model, technical report. *Groupe Intelligence Artificielle, université Aix- Marseille*, October 1982.

FAGES F., Programmation logique par contraintes. *Éditions ellipses*, 1996.

GLOVER F., Tabu search –part 1, *ORSA Journal on Computing*, vol.1, No. 3, 1986, pp.190-206.

HEDIBLE C., PIERRE S., Algorithme génétique pour l'affectation de cellules à des commutateurs, mémoire de maîtrise, *Dpt. de génie électrique et génie informatique, École Polytechnique de Montréal,* Novembre 2000.

HOLLAND J. H., Adaptation in Natural and Artificial System, *The University of Michigan Press, Ann Arbor*, Michigan 1975.

HOLLAND J. H., Genetic Algorithms and the optimal allocation of trials, *SIAM Journal of Computing*, Vol. 2, No. 2, 1973, pp. 88-105.

HOUETO F., PIERRE S., Affectation de cellules à des commutateurs dans les réseaux cellulaires mobiles, *article soumis aux Annales des Télécommunications*, 2000.

JAFFAR J., LASSEZ J.-L., Constraint logic programming. In *Proceedings of the 14th ACM Symposium on principles of Programming Languages,* pp111-119, Munich, Germany, January 1987. ACM Press.

JAFFAR J., MAHER M., Constraint logic programming: A survey. *Journal of Logic Programming,* 19é20:503-582, 1994.

KLINCEWICZ J. G., Heuristics for the p-hub location problem, *European Journal of Operation Research,* vol.53, 1991, pp.25-37.

MARRIOTT K. et STUCKEY P., *Programming with Constraints: an Introduction*, MIT Press, 1998.

MERCHANT A., SENGUPTA B., Assignment of Cells to Switches in PCS Networks, *IEEE/ACM Transactions on Networking,* vol.3, No.5,1995, pp.521-526.

MERCHANT A., SENGUPTA B., Multiway graph partitioning with applications to PCS Networks, *IEEE Infocom'94,* vol.2, 1994, pp.593-600.

PESANT G., An optimal algorithm for the traveling salesman problem with time windows using constraint logic programming, *Publication CRT (Centre de Recherche sur les Transports, Montréal)*, 1996, No 1030.

SAHA D., MUKHERJEE A. et BHATTACHARYA P. S., a simple heuristics for assignment of cells to switches in a PCS Networks, *Wireless Personal Communications*, vol12, 2000.

SAMADI B. et WONG W. S., Optimization Techniques for Location Area Partitioning, *8th ITC Specialist Sem. UPC*, Geneva, 1992.

SKORIN-KAPOV J., Tabu search applied to the quadratic assignment problem, *ORSA Journal on Computing*, vol. 2, No. 1, 1989, pp 33-45.

SUTHERLAND I., Sketchpad, a man-machine graphical communication system. *In proceedings of the Spring Joint Computer Conference*, pp 329-346. IFIPS, 1963.

VAN HENTENRYCK P., DEVILLE Y. et MICHEL L., Numerica. A modeling language for global optimization. *MIT Press*, 1997.